View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Thèses en ligne de l'Université Toulouse III - Paul Sabatier

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *l'Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier)*

Présentée et soutenue le *24/06/2013* **par :**

TOURIA CHAFQANE BEN RAHHOU

Nouvelles Méthodes pour les Problèmes d'Ordonnancement Cyclique.

JURY

AZIZ MOUKRIM	Université de Technologie de Compiègne	Président du Jury
SÉBASTIEN LAHAYE	Université d'Angers	Rapporteur
RÉMY DUPAS	Université de Bordeaux	Rapporteur
CYRIL BRIAND	Université Toulouse III	Examinateur
CHRISTIAN ARTIGUES	LAAS-CNRS	Examinateur
LAURENT HOUSSIN	Université Toulouse III	Directeur

École doctorale et spécialité :
EDSYS : Génie Industriel 4200046
Unité de Recherche :
Laboratoire d'Analyse et d'Architecture des Systèmes
Directeur de Thèse :
Laurent HOUSSIN
Rapporteurs :
Sébastien LAHAYE et *Rémy* DUPAS

Université Toulouse III - Paul Sabatier
118 Route de Narbonne
31 062 Toulouse cedex 9

Ĺaboratoire d'Analyse et d'Architecture des Systèmes
7, avenue du Colonel Roche
31 077 Toulouse Cedex 4

A mes parents,
A mes enfants,

Remerciements

Le présent travail n'aurait pas abouti sans le bienveillant soutien de certaines personnes. Je voudrais les prier d'accueillir ici tous mes sentiments de gratitude.

J'exprime tout d'abord mes profonds remerciements à mon directeur de thèse, Laurent Houssin pour l'aide compétente qu'il m'a apportée, pour sa patience, sa rigueur et sa disponibilité.

Merci à Christian Artigues pour son aide précieuse, son encouragement à finir ce travail et sa grande compétence pour le travail de relecture et de correction.

J'adresse aussi tous mes remerciements à Monsieur Rémy Dupas et Monsieur Sébastien Lahaye qui ont bien voulu être les rapporteurs du présent travail, ainsi qu'à Monsieur Aziz Moukrim pour l'honneur qu'ils me font d'accepter d'être membres du jury de cette thèse. Je les remercie pour l'attention qu'ils porteront à ce document et pour leur participation à sa soutenance.

Je souhaite exprimer ma reconnaissance à Cyril Briand ainsi qu'à tous les membres de l'équipe MOGISA pour leur disponibilité, et pour m'avoir guidée dans mon parcours de doctorante.

Enfin c'est à mon cher Saïd, pour son soutien, son indispensable coup de mains et ses encouragements renouvelés dans des moments difficiles que je veux exprimer toute mon affection et ma reconnaissance.

Résumé

Les travaux de recherche concernant l'ordonnancement mobilisent un nombre important de chercheurs. Cette forte émulation est principalement due au large panorama des problématiques d'ordonnancement. Parmi elles, le problème d'atelier à cheminement multiple, communément appelé « Job-Shop », tient une place particulièrement prépondérante tant ce problème est rencontré dans le milieu industriel. De nombreux sujets de recherche, en France et à l'étranger, sont issue de cette problématique.

Les problèmes de Job-Shop peuvent souvent être simplifiés en les considérant comme des problèmes cycliques. L'ordonnancement des tâches devient ainsi cyclique et son objectif est d'organiser les activités de production en répétant un cycle de base que l'on a optimisé. De nombreux paramètres entrent en jeu dans l'optimisation du cycle de base tels que la période du cycle choisie, l'ordre des opérations élémentaires pour réaliser un travail, la durée de ces opérations, le nombre de produits à réaliser par cycle, etc.

Plusieurs approches ont été utilisées pour résoudre ce problème. Parmi elles, nous pouvons citer l'approche par réseaux de Petri et plus particulièrement par graphes d'événements temporisés, l'approche par les graphes, l'approche par la programmation linéaire et l'approche par la théorie des tas.

L'approche par les graphes permet une représentation graphique du problème sous forme d'un graphe où les noeuds représentent les différentes opérations et où les arcs illustrent les contraintes du problème d'ordonnancement cyclique, un tel problème admet une solution réalisable si, et seulement si, le graphe associé est consistant. Cette propriété de consistance d'un problème d'ordonnancement cyclique et de son graphe permet d'élaguer l'arbre de recherche de la procédure de séparation et d'évaluation proposée pour cette approche.

Concernant l'approche par la théorie des tas, le sous-problème de l'évaluation d'une solution peut être résolu aisément avec l'aide de la théorie des tas. En effet, en traduisant le problème dans une structure mathématique adaptée, l'évaluation du taux de production du cycle revient au calcul d'une valeur propre d'un produit de matrices dans lequel chacune des matrices représente une opération élémentaire. Cette propriété s'avère particulièrement intéressante dans le cas de l'évaluation successive d'un grand nombre d'ordonnancement.

En outre, la théorie des tas permet une représentation très intuitive d'un ordonnancement, puisque celui-ci s'illustre comme un empilement de plusieurs briques (en fait, un « tas » de briques) dont le contour supérieur correspond aux dates de fin des dernières opérations des machines.

Mots-clefs

Ordonnancement cyclique, Algèbre max-plus, Optimisation discrète, Théorie des tas, Théorie des graphes.

Synthesis Algorithms for Cyclic Scheduling Problems

Abstract

Research on scheduling has been mobilizing a large number of researchers, this is mainly due to the broad range of application of scheduling problems, among them, there is the problem of tracking multiple workshops, commonly called "JobShop". JobShop problems can often be simplified by considering them as cyclic problems. The tasks scheduling becomes cyclic and its purpose is to organize the production activities by repeating a basic cycle that has been optimized. Many parameters are involved in the optimization of the basic cycle such as the period of the cycle chosen, the order of operations to perform basic work, the duration of these operations, the number of outputs per cycle, etc.

Several approaches have been used to solve this problem, among which the approach by Petri nets, especially in timed event graphs, graph approach, linear programming approach and heap of pieces approach.

The graph approach allows a graphical representation of the problem as a graph where the nodes represent the different operations and where the arcs illustrate constraints. A cyclic scheduling problem has a feasible solution if and only if it is consistent. This consistency property of a cyclic scheduling problem and its associated graph allows to prune the search tree of the branch-bound procedure proposed for this approach.

In the heap of pieces approach, the sub-problem of evaluating a solution can be easily solved. Indeed, by translating the problem into an appropriate mathematical structure, finding the cycle time is equivalent to the calculation of an eigenvalue of a product of matrices where each matrix represents an elementary operation. This property is particularly significant in the case of successive sets of evaluation of many schedules. In addition, the theory allows an intuitive representation of scheduling, since it is illustrated as a stack of several blocks. Several cycles will be symbolized by the stack of piles.

Keywords

Cyclic scheduling, Max-plus algebra, Discrete optimization, Heap of pieces, Graph theory.

Table des matières

Introduction

Contexte

Une organisation temporelle efficace de la production est une question cruciale pour la performance d'une entreprise. En effet, la fonction d'ordonnancement joue un rôle essentiel dans la gestion d'un système de production industrielle pour respecter les dates de livraison tout en minimisant les délais de production. Dans un contexte de compétitivité accrue, les entreprises sont ainsi incitées à développer un systhème d'allocation optimale des différentes parties de leurs ateliers de production.

La recherche opérationnelle a toujours été très active sur ces problèmes d'ordonnancement, qui selon cette discipline, appartiennent à une classe de problèmes mathématiques NP-complets marquée par l'explosion combinatoire. Les méthodes de résolution mathématique sont, en général :

- des méthodes exactes qui ont pour objectif de trouver des solutions optimales à partir d'algorithmes mathématiques complexes, mais qui souvent nécessitent des temps de calcul importants.
- des méthodes approximatives, qui se limite à des solutions réalisables (pas forcément optimales). Des régles de priorité sont établies pour construire ces heuristiques qui permettent des temps de calcul plus raisonnables.

Les problèmes d'ordonnancement sont étudiés dans une multitude de situations. L'organisation opérationnelle de la production dans les ateliers reste néanmoins un cas typique et fait l'objet de nombreux travaux. Quoique récente (début des années 1960), la théorie de l'ordonnancement a fait l'objet d'études très poussées, tant au niveau de la complexité, que de la recherche de solutions exactes ou de solutions approchées.

Problématique et contributions

Dans ce travail, nous nous intéressons au problème d'ordonnancement cyclique. En particulier, au problème de Job-Shop Cyclique en considérant des méthodes de résolution exactes. Peu de méthodes de résolution de ce genre de problèmes sont proposées dans la littérature. Cela est d'autant plus vrai pour les méthodes exactes.

Nous proposons une nouvelle méthodes de résolution exacte pour le problème dU Job-Shop Cyclique, cette méthode qui est une procédure de séparation et d'évaluation basée sur la théorie des graphe repose sur deux piliers majeurs :

- La consistance d'un graphe pour obtenir des bornes sur les décalages événementiels ;
- L'algorithme de Howard pour calculer la performance d'un ordonnancement.

Le Work-In-Process est le nombre de travaux en cours dans un même cycle, cette valeur représente le nombre d'en-cours dans un atelier de production. Toujours dans la recherche d'une meilleure compétitivité, les entreprises visent à minimiser les en-cours. Cependant, le fait d'augmenter les en-cours permet d'influencer le temps de cycle optimal et éventuellement de le réduire. Une borne supérieure pour le Work-In-Process est ainsi proposée pour l'optimalité d'un ordonnancement cyclique.

Dans une autre approche, l'approche par la théorie des tas, nous proposons également une procédure de séparation et d'évaluation pour les ordonnancements 1-cycliques. Dans une première adaptation de cette approche au problème du Job-Shop Cyclique, l'ensemble des solutions recherchées dans le cas d'ordonnancement K-cyclique est réduit à des ordonnancements où, à tout moment, il ne peut-y avoir qu'une seule occurrence en cours d'un travail donné. Dans le cadre de notre étude, nous proposons une extention de cet ensemble de solutions au cas général du problème de Job-Shop Cyclique.

Plan du mémoire

Ce mémoire est articulé en cinq chapitres. Les deux premiers chapitres introduisent les notions nécessaires à la compréhension de nos travaux et les trois derniers chapitres présentent nos contributions et résultats.

- Dans le premier chapitre, nous définissons le socle théorique. Les principales notions utiles au travail présenté dans ce mémoire sont introduites. Deux formalismes de modélisation des problèmes d'ordonnancement cyclique sont d'abord présentés : la théorie des graphes et les systèmes à événements discrets. Ensuite, deux modèles emergent de ce dernier formalisme, qui sont l'algèbre des dioïdes et les réseaux de Petri, sont rappelés. Enfin, une procédure d'optimisation progressive : la méthode de séparation et d'évaluation est elle aussi décrite.
- Dans le deuxième chapitre, un état de l'art des principaux problèmes d'ordonnancement cyclique est présenté. Ce chapitre bibliographique aborde en premier lieu le problème d'ordonnancement cyclique de base et l'ordonnancement K-cyclique pour proposer par la suite un survol des différents problèmes d'ordonnancement cyclique présents dans la littérature. En particulier, le problème de Job-Shop Cyclique qui fait l'objet de ce travail. Les différentes approches de modélisation et de résolution du problème de Job-Shop Cyclique sont ensuite listées.
- Dans le troisième chapitre, une nouvelle procédure de résolution exacte du problème de Job-Shop Cyclique est énoncée. Nous présentons tout d'abord les concepts théoriques nécessaires au développement de cette procédure de séparation et d'évaluation basée sur la théorie des graphes. La structure de l'algorithme est ensuite décrite. Ce chapitre est aussi l'occasion de rappeler quelques méthodes de résolution exacte de ce genre de problèmes, ces méthodes aux quelles nous comparons cette nouvelle procédure.
- Dans le quatrième chapitre, est présentée, la théorie des tas pour la résolution des problèmes de Job-Shop Cycliques. Cette approche, nouvelle et originale, consiste à modéliser un ordonnancement par un tas de pièces qui s'empilent. En traduisant le

problème dans une structure mathématique adaptée, à savoir, l'algèbre des dioïdes présentée dans le premier chapitre, l'évaluation du temps de cycle d'un ordonnancement donné se traduit par le calcul d'une valeur propre de matrice.

- Dans le cinqième chapitre, nous présenterons les expérimentations qui ont été menées au cours de cette thèse. Il s'agit d'une comparaison entre les différents modèles présentés dans ce mémoire en utilisant des instances générées aléatoirement et variant le type de solutions recherchées en fonction de la cyclicité de l'ordonnancement et du nombre de travaux en cours dans un cycle de production.

Une conclusion vient ensuite lister les principaux apports de ce travail de thèse et nous permettra de tracer quelques perspectives de recherche.

Notations

CJSP	Problème de Job-Shop cyclique
SED	Systèmes à événements discrets
RdP	Réseau de Petri
GET	Graphes d'événements temporisés
PSE	Procédure de séparation et d'évaluation
GBCSP	Problème d'ordonnancement cyclique de base
PSIP	Problème d'ordonnancement cyclique à machines parallèles
WIP	Work-In-Process
α	Le temps de cycle asymptotique d'un ordonnancement
τ	Le taux de production d'un ordonnancement
$\mathcal{T}$	L'ensemble des tâches d'un problème d'ordonnancement
$\mathcal{J}$	L'ensemble des travaux d'un problème d'ordonnancement
$\mathcal{R}$	L'ensemble des machines d'un problème d'ordonnancement
$G = (V, E)$	Un graphe G dont l'ensemble des noeuds est V et l'ensemble des arcs est E
(i, j)	Un arcs orientés allant du noeud i au noeud j
$t(i, k)$	La date de début de la $k^{ème}$ occurrence de la tâche i ($k \in \mathbb{N}$)
p_i	La durée de la tâche i
$< i, k >$	La $k^{ème}$ occurrence de la tâche i ($k \in \mathbb{N}$)
H_{ij}	La hauteur entre les occurrences des tâches i et j ($H_{ij} \in \mathbb{Z}$)
K_{ij}	Le décalage événementiel entre les occurrences des tâches i et j ($K_{ij} \in \mathbb{Z}$)
L_{ij}	La longeur de la précédence entre les occurrences des tâches i et j ($L_{ij} \in \mathbb{R}^+$)
$\mathbb{R}_{max}$	L'ensemble des réels $\mathbb{R}$ augmenté de $-\infty$
$\oplus$	La loi additive dans l'algèbre $(max, +)$, correspond à l'opération max
$\otimes$	La loi multiplicative dans l'algèbre $(max, +)$, correspond à l'opération max
ε	L'élément neutre pour $\oplus$, $\varepsilon = -\infty$
e	L'élément neutre pour $\otimes$, $e = 0$

Chapitre 1

Outils préliminaires

La théorie de l'ordonnancement a souvent fait appel à des modèles et des techniques de résolutions différentes et provenant de communauté distincte. Ce premier chapitre est donc l'occasion de rassembler un certain nombre de notions utiles à la compréhension des problèmes d'ordonnancement traités dans ce document. En outre, nous introduisons ici le vocabulaire et les notations utilisés par la suite.

La première partie de ce chapitre est consacrée aux rappels ce quelques notions de théorie des graphes, celle-ci étant très souvent à la base de modèles pour les problèmes d'ordonnancement (cycliques ou non cycliques). Préférablement employés par la communauté "automaticienne", les systèmes à événements discrets peuvent être utilisés pour la modélisation des problèmes d'ordonnancement cyclique, ils font l'objet d'une seconde partie. Deux modèles emergent de cette dernière catégorie : l'algèbre $(max, +)$ et les réseaux de Petri. L'algèbre $(max, +)$, aussi appellé "algèbre des chemins" à cause de ses fortes connexions avec la théorie des graphes, présente un intérêt particulier dans la modélisation des systèmes à événements discrets mettant en jeu des phénomènes de synchronisation mais sans phénomène de concurrence (c'est le cas d'une solution d'ordonnancement). De plus, les matrices à coefficients dans l'algèbre $(max, +)$ présentent certaines propriétés intéressantes dans le cadre de l'étude des systèmes dynamiques à événements discrets. On formule quelques rappels sur cette structure algébrique dans la troisième partie. Dans une quatrième section, on présente brièvement les réseaux de Pétri et plus particulièrement les graphes d'événements temporisés. Cette sous classe de réseaux de Petri est souvent utilisée pour modéliser, au moins graphiquement, des solutions d'ordonnancement cycliques. Enfin, la dernière partie de ce chapitre est consacrée à un outil classique de résolution de problème d'optimisation discret : la procédure de séparation et d'évaluation. Cette méthode est très souvent à la base de méthode de résolution de problème d'ordonnancement.

1.1 La théorie des graphes

La théorie des graphes est apparue pour la première fois en 1735 grâce au mathématicien Leonhard Euler, mais ce n'est qu'en 1960, que les bases de la théorie moderne des graphes ont été posées. Dans cette partie, nous allons définir les principales définitions et notations relatives aux graphes. Pour plus de détails sur ces notions, le lecteur pourra se

référer à de nombreux ouvrages dont [Ber70, Die10, GM79].

Définition 1.1. Un graphe non orienté G est la donnée d'un couple (V_G, E_G), où V_G est un ensemble de $n \neq 0$ noeuds, et E_G un ensemble de $m \neq 0$ arcs formés de paires de noeuds distincts.

Un arc entre les noeuds u et v est noté $(u, v) \in E_G$. Deux noeuds $u, v \in V_G$ sont dits adjacents, s'il existe un arc $e = (u, v) \in E_G$. Dans ce cas, chacun de ces deux noeuds u et v est dit incident à l'arc e. Deux arcs $e, f \in E_G$ sont dits adjacents s'ils partagent un noeud incident commun v, c'est-à-dire $e = (u, v)$ et $f = (v, w)$ avec $u, v, w \in V_G$.

- Le voisinage ouvert d'un noeud $v \in V_G$, noté $\mathcal{N}_G(v)$, est l'ensemble des noeuds adjacents à v : $\mathcal{N}_G(v) = \{u|(u, v) \in E_G\}$.
- Le voisinage fermé de v, note $\mathcal{N}_G[v]$, inclut également le noeud v lui-même : $\mathcal{N}_G[v] = \mathcal{N}_G(v) \cup v$.
- Le voisinage ouvert d'un sous-ensemble de noeuds $S \subset V_G$, noté $\mathcal{N}_G(S)$, est l'ensemble des noeuds de V_G ayant un voisin dans S : $\mathcal{N}_G(S) = \{u \in V_G|\exists v \in S : (u, v) \in E_G\}$.
- Le voisinage fermé de S est alors $\mathcal{N}_G[S] = \{\mathcal{N}_G(S) \cup S\}$.
- Le degré d'un noeud $v \in V_G$, noté $d_G(v)$, est égal au nombre de voisins qu'il a dans G : $d_G(v) = |\mathcal{N}(v)|$.
- Le degré maximum de G, noté $\Delta(G)$, est alors defini par : $\Delta(G) = max_{v \in V_G} \{d_G(v)\}$.
- Un noeud est dit isolé s'il est de degré 0, et n'est donc voisin avec aucun autre noeud du graphe.
- Un arc est dit isolé si ses deux noeuds incidents sont de degré 1, et ont donc comme seul voisin l'autre sommet incident à ce même arc.

Définition 1.2. Un chemin P dans un graphe G est une suite $(v_1, v_2, \ldots, v_k)$ de noeuds dans V_G, telle que $(v_i, v_{i+1}) \in E_G$ pour tout $1 \leq i \leq k - 1$. La longueur d'un chemin est égal au nombre d'arcs qui le composent. Un chemin est dit élémentaire s'il ne passe qu'une seule fois par le même noeud, autrement dit, $v_i \neq v_j$ pour tout $1 \leq i \neq j \leq k - 1$.

Un cycle C est un chemin élémentaire revenant à son noeud de depart, *i.e.* $v_1 = v_k$.

Définition 1.3. Un graphe G est *fortement connexe* s'il existe un chemin entre toute paire de noeuds du graphe.

Un graphe valué est un graphe dans lequel chacun des arcs présente une valeur. Cette valeur peut représenter une distance, un temps, un montant d'argent, *etc.*

Définition 1.4. Un graphe (orienté ou non) $G = (V_G, E_G)$ est valué s'il est muni d'une application

$$v : E_G \to \mathbb{R}$$

$$(x, y) \mapsto v(x, y)$$

qui est appelée *valuation*.

Définition 1.5. Soit $G = (V_G, E_G, v)$ un graphe valué. La valuation d'un chemin est la somme des valuations de chacun des arcs qui le composent.

Nous pouvons maintenant définir plus court chemin entre deux noeud d'un graphe G :

Définition 1.6. Soit $G = (V_G, E_G, v)$ un graphe valué et soient u, w deux noeuds de G.
- On appelle distance de u à w et on note $l(u, w)$ le minimum des valuations des chemins allant de u à w.
- On appelle plus court chemin de u à w tout chemin dont la valuation est égale à $l(u, w)$.

De nombreux problèmes concrets peuvent se modéliser comme des recherches de plus courts chemins dans des graphes valués. Par exemple : la recherche de l'itinéraire le plus rapide en voiture entre deux villes, le routage dans des réseaux de télécommunications et certains problèmes d'ordonnancement font aussi appel à des recherches de plus courts chemins.

1.2 Systèmes à événements discrets

Les "systèmes à événements discrets" (SED) recouvrent des systèmes dynamiques qui s'intéressent uniquement aux "événements disctrets", autrement dit, les SEDs ne prennent en compte que le début et la fin de certains phénomènes. Ces systèmes qui sont utilisés pour modéliser des phénomènes d'organisation, de synchronisation et de coopération, trouvent leurs applications dans plusieurs domaines.

Un premier exemple est la modélisation de processus de fabrication de systèmes industriels manufacturiers. On définit un travail comme une suite de tâches effectuées dans un ordre bien déterminé. L'industrie manufacturière est caractérisée par la production simultanée de petites et moyennes séries en respectant un ratios de production, par exemple la production de véhicules. Les variables discrètes du système représentent les nombres de pièces. Un travail utilise des ressources : les machines qui effectuent les différentes tâches, les stocks, les convoyeurs entre les différentes machines, etc. Il apparaît alors des problèmes de synchronisation et de coopération entre des travaux s'effectuant en parallèle et partageant les mêmes ressources. Ces problèmes ont une acuité de plus en plus forte, il est donc très important de les aborder par des méthodes efficaces pour garantir un fonctionnement efficace des systèmes.

Les systèmes informatiques, par exemple un réseau d'ordinateurs communiquant par satellites, peuvent aussi être modélisés par des SED puisqu'eux aussi effectuent des transformations sur de l'information. On peut définir un processus informatique de façon similaire à un travail manufacturier utilisant des ressources : processeurs pour transformer l'information, mémoires pour la stocker, réseaux pour la transporter, etc. Les systèmes informatiques, du fait de l'augmentation de leur complexité et de leurs interconnexions, sont très difficiles à gérer, sans méthodes particulièrement efficaces.

D'autres exemples peuvent être cités comme l'organisation du traffic dans des réseaux de communication ou dans des réseaux routiers, ou encore la gestion d'un projet...

De façon générale, les SED présentent les caractéristiques suivantes :
- Le parallélisme : certains événements se déroulent simultanément et indépendamment dans divers parties du système.

– La synchronisation : la disponibilité simultanée des ressources utiles et la vérification simultanée des conditions requises pour chaque événement sont nécessaires à la réalisation de celui-ci.
– La concurrence : certains événements partageant des ressources limitées ne peuvent s'excuter simultanément.

Nous allons citer, dans ce qui suit, quelques approches usuelles dans l'études des SED.

1.3 Algèbre des dioïdes

1.3.1 Présentation de l'algèbre *(max,+)*

Les systèmes à événements discrets qui mettent en jeu des phénomènes de synchronisation ne peuvent pas être décrits dans l'algèbre usuelle par des équations linéaires à cause de la non linéarité de l'opérateur " max ". L'algèbre des dioïdes appelé, aussi l'algèbre des chemins ou encore l'algèbre $(max, +)$ est une algèbre particulière qui permet la modélisation de ces systèmes.

L'algèbre $(max, +)$ est un formalisme mathématique, particulièrement adapté pour décrire les phénomènes de parallélisme, retard et synchronisation. Une théorie a d'ailleurs été développée pour les systèmes mettant en jeu ces phénomènes [BCOQ92]. A partir de cette théorie des travaux de commande ont été élaborés pour la classe des systèmes concernés, on peut citer sans exhaustivité [HLB12] et [KLB09]. Les applications se rencontrent régulièrement dans les systèmes de production (et notamment l'ordonnancement) [CDQV83] [BLB06], les systèmes de transport [Bra93] [LMQ05] [HLB04] et les systèmes informatiques [BP01].

Définition 1.7. L'algèbre $(max, +)$ est l'ensemble $\mathbb{R}_{max} = \mathbb{R} \cup \{-\infty\}$ muni de deux opérations associatives notées $\oplus$ et $\otimes$ telles que $\oplus$ est commutative et $\otimes$ est distributive par rapport à $\oplus$ avec les deux propriétés suivantes :

$\forall a \in \mathbb{R}_{max}$

– le zéro ε est absorbant pour la multiplication : $\varepsilon \otimes a = a \otimes \varepsilon = \varepsilon$;
– l'addition est idempotente : $a \oplus a = a$;

Dans l'algèbre $(max, +)$, la loi additive $\oplus$ correspond à l'opération max, et la loi multiplicative $\otimes$ désigne la somme usuelle. De plus l'élément neutre $e = 0$ est neutre pour le produit :

$$\forall a \in \mathbb{R}_{max}, \quad a \otimes e = a.$$

Une présentation exhaustive de cette structure mathématique est faite dans [BCOQ92]. Plus particulièrement, on considère l'algèbre $(max, +)$ matricielle obtenue en considérant les matrices carrées de taille n à coefficient dans $\mathbb{R}_{max}$ et en munissant cet ensemble de la somme et du produit matriciels usuels :

Définition 1.8. Pour $\mathrm{A} = (A_{ij})$ et $\mathrm{B} = (B_{ij})$ deux matrices carrées de taille n, on définit la somme et le produit matriciels comme suit :

– $(A \oplus B)_{ij} = A_{ij} \oplus B_{ij}$;
– $(A \otimes B)_{ij} = \bigoplus_{k=1}^{n} (A_{ik} \otimes B_{kj})$.

Les matrices carrées à coefficients dans un dioïde présentent certaines propriétés spectrales. Ces propriétés peuvent être illustrées par la théorie des graphes. Il existe en effet une correspondance directe entre les manipulations combinatoires de matrices définies dans $\mathbb{R}_{max}$ et le cheminement dans leurs graphes associés.

On rappelle ici un ensemble de résultats permettant de caractériser le spectre, c'est-à-dire l'ensemble des valeurs propres, des matrices carrées à coefficients dans $(max, +)$.

La propriété de "cyclicité" de telles matrices est mise en avant. En effet, cette propriété est utile pour caractériser le comportement asymptotique des systèmes $(max, +)$ linéaires, en permettant précisément de mettre à jour l'existence d'un régime périodique.

Définition 1.9 (Graphe de précédence)**.** Pour une matrice carrée $A \in \mathcal{D}^{n\times n}$, il existe un graphe nommé graphe de précédence, noté $\mathcal{G}(A)$ et composé de n noeuds. L'arc (j, i) de ce graphe est pondéré par l'élément A_{ij} si $A_{ij} \neq \varepsilon$, sinon l'arc (j, i) n'existe pas. Inversement à tout graphe composé de n noeuds, on peut associer une matrice de précédence de dimension $n \times n$.

Théorème 1.10. *Soit A une matrice carrée à coefficients dans $(max, +)$. Dans le graphe de précédence associé à la matrice A, $(A^k)_{ij}$ est égal à la somme (i.e. au max dans l'algèbre usuelle) des poids des chemins de longueur k allant de j à i.*

Définition 1.11 (Matrice irréductible)**.** Une matrice $A \in \mathcal{D}^{n\times n}$ est dite irréductible si pour toute paire (i, j), il existe un entier m tel que $(A^m)_{ij} \neq \varepsilon$.

En se référant à la définition 1.10, l'irréductibilité d'une matrice peut être interprétée en affirmant que, pour toute paire de noeuds (i, j) de son graphe de précédence, il existe un chemin de j à i.

Théorème 1.12. *Soit A une matrice carrée à coefficients dans $\mathbb{R}_{max}$. Les deux affirmations suivantes sont équivalentes.*
(i) La matrice A est irréductible.
(ii) Le graphe de précédence associé à la matrice A est fortement connexe.

On entend par valeur propres et vecteurs propres d'une matrice $A \in \mathcal{D}^{n\times n}$, les scalaires $\lambda \in \mathcal{D}$ et les vecteurs $v \in \mathcal{D}^n \setminus \varepsilon$ tels que

$$A \otimes v = \lambda \otimes v.$$

Théorème 1.13. *Soit A une matrice carrée à coefficients dans $(max, +)$ et $\mathcal{G}(A)$ le graphe de précédence qui lui est associé. Si A est irréductible, ou de façon équivalente si $\mathcal{G}(A)$ est fortement connexe, alors il existe une unique valeur propre notée λ. Cette valeur propre est égale au poids moyen maximum du graphe, c'est-à-dire le maximum des poids moyens des circuits du graphe.*

Notons qu'à l'unique valeur propre d'une matrice irréductible peuvent être associés plusieurs vecteurs propres.

Définition 1.14 (Circuit critique)**.** Soit une matrice A irréductible de valeur propre λ et $\mathcal{G}(A)$ le graphe qui lui est associé. Le circuit ζ de $\mathcal{G}(A)$ est dit critique si son poids moyen est maximum, c'est-à-dire égale à λ.

Le théorème suivant introduit la propriété de cyclicité des matrices à éléments dans un dioïde.

Théorème 1.15 (Cyclicité)**.** *Pour une matrice irréductible $A \in \mathcal{D}^{n\times n}$ de valeur propre λ, il existe deux entiers N et c tels que*

$$\forall n \geq N \ , \ A^{n+c} = \lambda^c \otimes A^n \ .$$

L'entier c est appelé cyclicité de A.

L'entier N indique l'entrée dans le régime périodique. Cette dernière propriété est utile dans l'analyse des systèmes périodiques.

1.3.2 Aspect Calculatoire

Le théorème 1.13 laisse entrevoir une méthode naïve pour calculer la valeur propre d'une matrice irréductible (à partir de son graphe de précédence). En effet, en s'inspirant du fait que l'élément $(A^j)_{ii}$ représente le poids maximal de tous les circuits de longueur j comprenant le noeud i. L'expression $\bigoplus_{i=1}^{n}(A^j)_{ii}$ nous donne le poids maximal des circuits de longueur j. Il n'est pas utile de faire varier j au delà de n, car c'est équivalent en termes de graphes à parcourir des circuits déjà parcouru. En accord avec le théorème 1.13, le poids moyen maximum d'un graphe peut donc s'exprimer de la façon suivante :

$$\lambda = (\bigoplus_{j=1}^{n}(\bigoplus_{i=1}^{n}(A^j)_{ii}))^{1/j}. \tag{1.1}$$

Concernant les vecteurs propres, une méthode de calcul basée sur la connaissance de la valeur propre est énoncée ci-dessous.

Définition 1.16. Soit $A \in \mathcal{D}^{n\times n}$ une matrice irréductible de valeur propre $\lambda \in \mathcal{D}$. On définit la matrice notée A_λ par

$$A_\lambda = \lambda^{-1} \otimes A.$$

Théorème 1.17. *Soit $A \in \mathcal{D}^{n\times n}$ une matrice irréductible de valeur propre λ. La j-ième colonne de la matrice A_λ^+, notée $(A_\lambda^+)_{.j}$, est un vecteur propre de A si elle satisfait l'égalité*

$$(A_\lambda^+)_{.j} = A_\lambda \otimes (A_\lambda^+)_{.j} \ .$$

Le poids moyen maximum d'un graphe peut aussi être calculé à l'aide d'algorithmes bien plus performants que la méthode présentée en 1.1, l'algorithme de Karp et l'algorithme de Howard en sont deux exemples :

Algorithme de Karp

L'algorithme de Karp présenté dans le théorème qui suit permet un calcul en $O(n^3)$ [DG98].

Théorème 1.18 (Théorème de Karp)**.** *Pour une matrice irréductible $A \in \mathcal{D}^{n\times n}$, l'unique valeur propre est donné par*

$$\lambda = \max_{i=1,\ldots,n} \min_{k=1,\ldots,n-1} \frac{(A^n)_{ij} - (A^k)_{ij}}{n-k} \tag{1.2}$$

pour tout $j \in 1,\ldots,n$.

Algorithme de Howard

L'algorithme de Howard (voir [How60]) est l'algorithme choisi pour le calcul de temps de cycle dans ce travail de thèse. Initialement conçu pour les processus décisionnels de Markov, l'algorithme de Howard a été adapté à l'algèbre $(max,+)$ dans [CTCG$^+$98] pour la résolution de problème d'ordonnancement cyclique de base. Bien que la complexité de l'algorithme de Howard reste non prouvée, le choix de cet algorithme pour notre procédure est dû au fait qu'il est le plus rapide des algorithmes connus pour le calcul de temps de cycle d'un graphe, d'après Dasdan et Gupta [DG98]. Dans [CTCG$^+$98], les auteurs montrent que l'algorithme de Howard nécessite un temps de calcul presque linéaire. Un schéma de cet algorithme est représenté dans la figure 1.1.

L'algorithme de Howard est basée sur deux algorithmes connus en chaînes de Markov :

- le premier algorithme "Algorithme 1" consiste en une itération sur les valeurs : à partir d'une matrice polynomiale A et d'une politique π, cet algorithme fournit la valeur propre et le vecteur propre de la matrice A^π.
- le deuxième algorithme "Algorithme 2" fait une itération sur les politiques : à partir d'une matrice polynomiale A, d'une politique π, d'une valeur propre et d'un vecteur propre, cet algorithme calcule une meilleure politique.

L'idée général de l'algorithme de Howard est la même que dans tous les autres algorithmes basés sur l'algorithme de Karp et Orlin [KO81] (voir aussi [IS95], [LK98] et [NYO91]). L'algorithme commence avec un petit temps de cycle α puis il augmente, selon une méthode spécifique, le α jusqu'à ce que certaines conditions d'optimalité soient satisfaites.

L'algorithme est divisé en deux partie, la première permet d'initialiser le sous-graphe $G_\pi = (T, E_\pi)$ de G appelé *graphes de politique* où $E_\pi = \{(u, succ(u)) | u \in T\}$. Ce sous-graphe G_π a le même ensemble de noeuds que G, cependant chaque noeud u de G_π a un et un seul successeur $succ(u)$. La mise à jour du sous-graphes G_π se fait en mettant à jour la fonction $succ()$. Après l'initialisation, G_{pi} est composé de plusieurs circuits disjonctifs. Au début de la deuxième partie, G_π est toujours composé de plusieurs circuits disjonctifs, et pour tous les noeuds qui n'appartiennent pas à un circuit, il existe un chemin qui mène à un des noeuds de ce circuit. Ces circuits sont examinés pour vérifier la consistence du graphe G_π, si cette condition est remplie, l'algorithme peut déterminer le circuit C^* qui a la valeur maximale de α dans G_π. Par la suite, la graphe G_π est changé de façon à ce

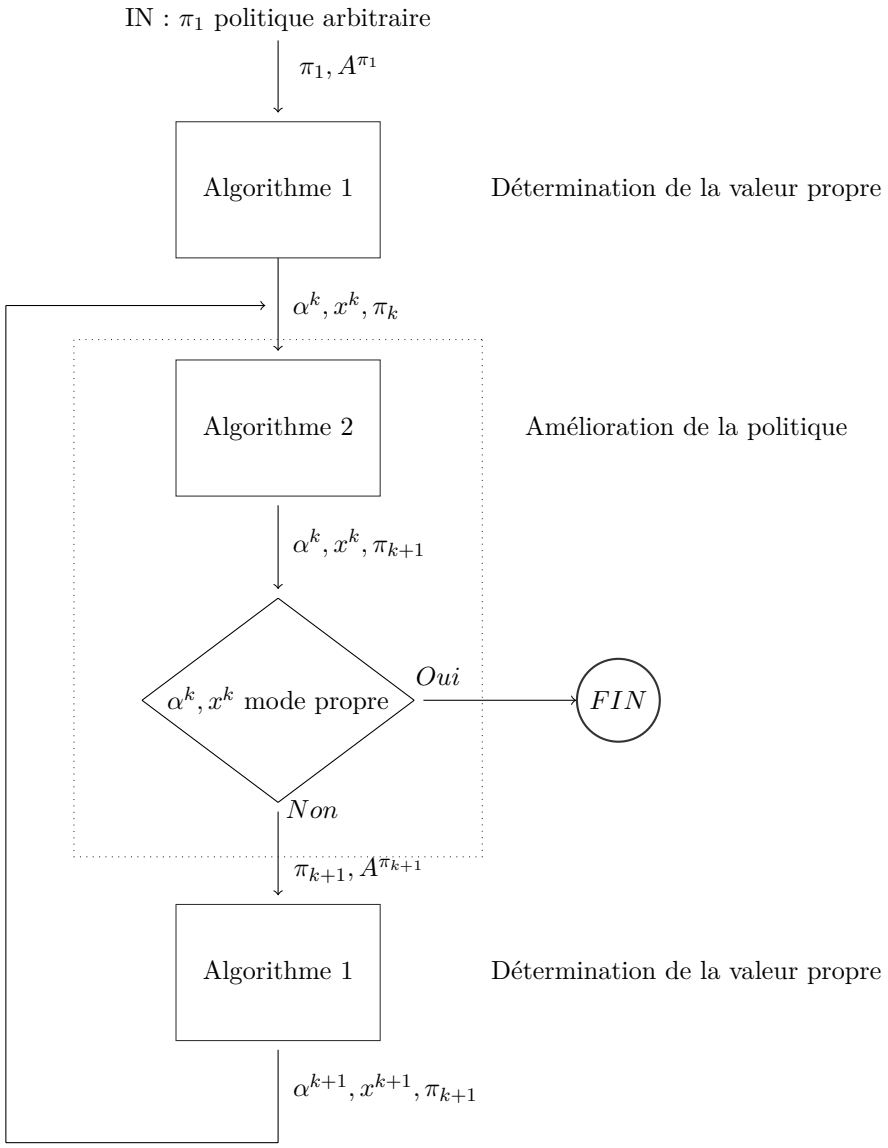

FIGURE 1.1 – Schéma de l'algorithme de Howard.

que C^* reste le seul circuit de G_π, et pour tous les autres noeuds qui n'appartiennent pas à ce circuit, il existe un chemin qui mène à un des noeuds du circuit C^*. En suite, les étiquettes $d(u)$ ($d(u)$ est une estimation de la valeur du plus long chemin qui part de u vers un noeud choisi s dans le circuit de valeur maximal C^*) sont calculer dans le circuit G_π. A la fin de la deuxième partie, l'algorithme vérifie si l'on peut améliorer les étiquettes $d(u)$ en changeant les arcs dans le graphe G_π, deux situations sont possibles :

- Les étiquettes $d(u)$ ne peuvent pas être améliorer, dans ce cas, l'algorithme quitte la boucle : il a trouvé le temps de cycle minimal.
- Les étiquettes $d(u)$ sont améliorées, dans ce cas, le nouveau graphe G_π est composé encore une fois de plusieurs circuits disjonctifs, et pour tous les noeuds qui n'appartiennent pas à un circuit, il existe encore une fois un chemin qui mène à un des noeuds de ce circuit. La boucle "While" recommence une nouvelle fois en examinant

tous les circuits du nouveau graphe G_π.

1.4 Réseaux de Petri et graphe d'événements temporisés

Les Réseaux de Petri ont été inventés par Carl Maria Petri au début des années soixante. Des travaux ultérieurs ont permis de développer les Réseaux de Petri comme un outil graphique de modélisation de certains phénomènes de synchronisation et de concurrence mais aussi comme un outil algébrique puisqu'il est possible de déduire de ce modèle une équation d'évolution et donc de le considérer comme un système dynamique à part entière. Les réseaux de Petri sont ici abordés pour leur intérêt à modéliser graphiquement les problèmes d'ordonnancement cyclique mais leur champs d'application est bien plus vaste et la littérature les concernant très fournies.

1.4.1 Définition d'un réseau de Petri

Un réseau de Petri (RdP) est un graphe orienté comprenant deux types de sommets :
- les places
- les transitions

Ils sont reliés par des arcs orientés . Un arc relie soit une place à une transition, soit une transition à une place mais jamais une place à une place ou une transition à une transition. Un exemple de Réseaux de Petri est représenté Figure 1.2.

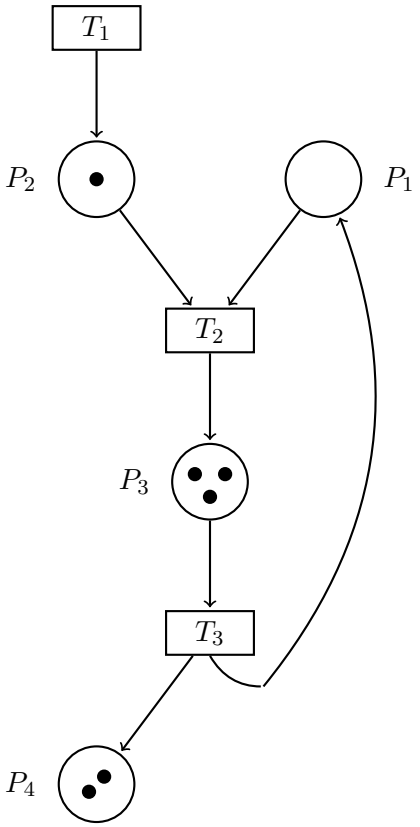

FIGURE 1.2 – Représentation d'un réseau de Petri.

Une place correspond à une variable d'état du système modélisé et une transition à un événement qui va entraîner l'évolution des variables d'état du système. A un instant donné, une place contient un nombre entier de jetons qui va évoluer en fonction du temps : il indique la valeur de la variable d'état à cet instant. Le nombre de jetons dans une place peut s'interpréter comme le nombre de ressources disponibles. L'absence de jeton indique qu'aucune ressource n'est disponible. Ces ressources sont consommées ou/et produites au cours du temps. Les jetons d'une même place n'ont pas d'identité individuelle.

On appelle marquage $\mathcal{M}$ d'un Réseau de Petri le vecteur du nombre de jetons dans chaque place : la $i^{ème}$ composante correspond au nombre de jetons dans la $i^{ème}$ place. Il indique à un instant donné l'état du RdP. Par exemple, le marquage du RdP présenté figure 1.2 est donné par :

$$\mathcal{M} = \begin{bmatrix} 0 \\ 1 \\ 3 \\ 2 \end{bmatrix}. \tag{1.3}$$

On peut maintenant définir plus formellement des RdPs (voir [Mur89], [DA89] pour plus de détails sur les RdPs).

Définition 1.19. On appelle Réseau de Petri non marqué le quadruplet $\mathcal{Q} = < \mathcal{P}, \mathcal{T}, \mathcal{I}, \mathcal{O} >$ où

- $\mathcal{P}$ est un ensemble fini non vide de places
- $\mathcal{T}$ est un ensemble fini non vide de transitions
- $\mathcal{P} \cap \mathcal{T} = \emptyset$
- $\mathcal{I}(\mathcal{T}_i)$ est l'ensemble des places qui sont en entrée de la transition i
- $\mathcal{O}(\mathcal{T}_i)$ est l'ensemble des places qui sont en sortie de la transition i.

On appelle Réseau de Petri marqué $\mathcal{R} = < \mathcal{Q}, \mathcal{M}_0 >$ où $\mathcal{M}_0$ est le marquage initial (le marquage à l'instant initial $t = 0$).

Définition 1.20. Une place $\mathcal{P}_i$ est bornée pour un marquage initial $\mathcal{M}_0$ si pour tout marquage accessible à partir de $\mathcal{M}_0$, le nombre de jetons dans $\mathcal{P}_i$ reste borné. Elle est dite k-bornée si le nombre de marques dans $\mathcal{P}_i$ est toujours inférieur ou égal à k. Un RdP marqué est k-borné si toutes ses places sont k-bornées.

On peut ainsi définir les réseaux de Petri saufs.

Définition 1.21. Un RdP marqué est sauf pour un marquage initial $\mathcal{M}_0$ s'il est 1-borné.

1.4.2 Graphes à événements temporisés

Les graphes d'événements temporisés (GET) sont une sous classe de réseaux de Petri. Dans un graphe d'événements, chaque place possède en amont une transition et en aval une transition, les conflits n'apparaissent donc jamais. Autrement dit, les graphes d'événements peuvent modéliser la synchronisation mais pas la concurrence (les phénomènes de concurrences se modélisent par des graphes d'état). Dans un GET, on associe à chaque

place un temps minimal de séjour des jetons. Tous les arcs d'un GET ont un poids unitaire. On suppose que les transitions sont franchies dès que toutes les places en amont sont marquées d'un jeton au moins (fonctionnement au plus tôt).

Pour l'étude des systèmes à événements discrets, les GET sont généralement utilisés comme outil de modélisation intermédiaire. Si la modélisation graphique constitue une première étape, la seconde étape est la mise en équations du modèle graphique, c'est-à-dire la définition d'une représentation algébrique du système.

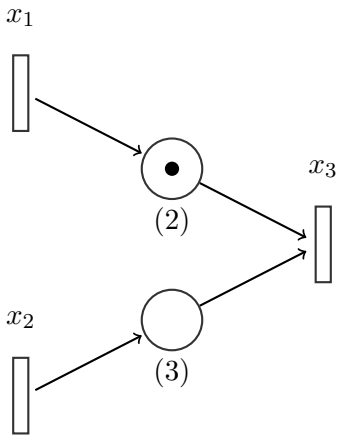

FIGURE 1.3 – Portion d'un graphe d'événements temporisés

On associe à chaque transition x une fonction appelée dateur $x(k)$ qui désigne la date du tir numérotée k de la transition x. Les dateurs sont définis sur le domaine événementiel car k est un numéro d'événement.

De façon générale, les dateurs sont des applications croissantes définies sur le domaine événementiel.

Le GET présenté en figure 1.3 nous indique que la date du k-ième franchissement (au plus tôt) de la transition x_3, notée $x_3(k)$, dépend alors :

- du $k-1$-ième franchissement (au plus tôt) de x_1 plus 2 unités de temps (soit $x_1(k-1)+2$)
- du k-ième franchissement (au plus tôt) de x_2 plus 3 unités de temps (soit $x_2(k)+3$)

La mise en équation du GET de la figure 1.3 en termes d'équations aux dateurs donne :

$$x_3(k) = max\ (x_1(k-1)+2,\ x_2(k)+3).$$

Dans l'algèbre $(max, +)$, l'équation devient

$$x_3(k) = 2x_1(k-1) \oplus 3x_2(k).$$

D'une manière générale, les équations modélisant le fonctionnement des GET ont la forme :

$$x(k) = \bigoplus_{i=0}^{a} A_i x(k-i) \oplus \bigoplus_{j=0}^{b} B_j u(k-j) \tag{1.4}$$

L'équation (1.4) est appelée "équation d'état" ou "équation d'évolution". Dans certains cas i peut prendre une valeure négative dans (1.4), le système est alors considéré comme non causal.

Certains systèmes $(max, +)$-linéaires possèdent la particularité d'être autonomes, c'est-à-dire qu'ils ne comportent pas d'entrée [1]. C'est le cas des systèmes de production cyclique. Leur évolution est donc uniquement décrite à partir des conditions initiales et de la matrice d'évolution. La représentation d'état de ces systèmes prend la forme suivante (lorsqu'ils sont causaux) :

$$x(k) = Ax(k-1). \tag{1.5}$$

En effectuant les itérations successives de la relation (1.5), on obtient

$$\begin{aligned} x(k) &= Ax(k-1) \\ &= A^2x(k-2) \\ &= \vdots \\ &= A^kx(0). \end{aligned}$$

Soit

$$x(k+c) = A^{k+c}x(0).$$

On peut remarquer que l'évolution de l'état peut être caractérisée simplement à partir des puissances de la matrice d'évolution. On distingue deux cas suivant la structure de la matrice d'évolution. Si celle-ci est irréductible, après un régime transitoire le vecteur d'état tout entier entre dans un régime périodique.

La périodicité de ces systèmes est directement déduite de la propriété de cyclicité des matrices (max,+) irréductibles (énoncée dans le théorème 1.15). Pour tout k suffisamment grand, on a en effet

$$\begin{aligned} x(k+c) &= A^{k+c}x(0) \\ &= \lambda^c A^k x(0) \\ &= \lambda^c x(k). \end{aligned}$$

Autrement dit, c occurrences d'événements ont lieu en $c \times \lambda$ unités de temps, où λ est la valeur propre de A et c sa cyclicité (*cf.* théorème 1.15). Le scalaire λ correspond par conséquent au temps de cycle moyen (soit l'inverse du taux de production).

1.5 La méthode de séparation et d'évaluation

La procédure de séparation et d'évaluation (PSE), également appelé selon le terme anglophone "Branch and Bound", est une méthode algorithmique classique pour résoudre un problème d'optimisation combinatoire. Elle permet de rechercher une solution optimale

1. ou encore que l'influence des entrées sur l'évolution de l'état peut être négligée (typiquement, $u(k) = \varepsilon$, $\forall k$).

dans un ensemble combinatoire de solutions possibles. La méthode repose d'abord sur la séparation (branch) de l'ensemble des solutions en sous-ensembles distincts. L'exploration de ces solutions utilise ensuite une évaluation optimiste (borne supérieure) pour majorer (bound) les sous-ensembles, ce qui permet de ne considérer que ceux susceptibles de contenir une solution meilleure que la solution courante.

Si E_0 est l'ensemble de solutions au problème, supposé discret et fini mais très grand, on énumère tous les éléments de E_0 en le séparant en un certain nombre de sous-ensembles non vides et disjoints, de taille variable. La figure 1.4 représente un exemple de découpage de l'ensemble de solution E_0.

On recommence la séparation en sous-ensembles avec chaque sous ensemble et ainsi de suite jusqu'à ce que tous les ensembles ne contiennent plus qu'un seul élément. Cette énumération peut se représenter par un arbre comme dans la figure 1.5 où la racine de l'arbre représente E_0, ses fils représentent les sous-ensembles créés dans la première partition de E_0, et ainsi de suite.

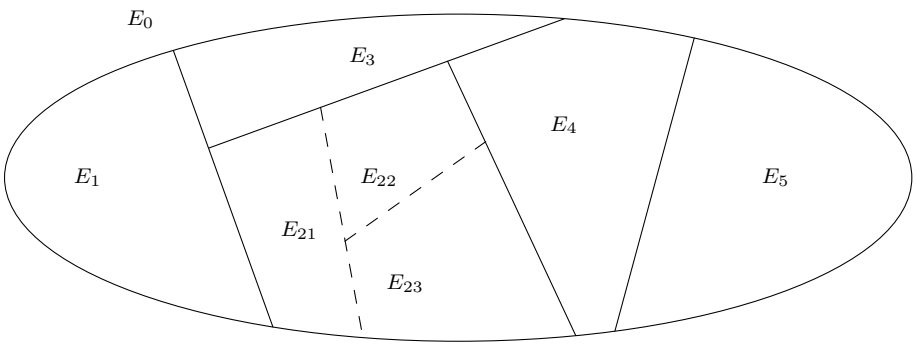

FIGURE 1.4 – Découpage d'ensemble dans une procédure de séparation et d'évaluation.

Le problème consiste à trouver dans un ensemble E_0 de solutions potentielles donné un élément de valeur optimale. Comme dans tous les problèmes d'optimisation, on cherche un élément de valeur optimale et non tous. L'algorithme propose de parcourir l'arborescence des solutions possibles en évaluant chaque sous-ensemble de solutions de façon optimiste. Lors de ce parcours, il maintient la valeur BS de la meilleure solution trouvée jusqu'à présent. Quand l'évaluation d'un sous-ensemble donne une valeur plus grande que BS, dans un problème de minimisation, il est inutile d'explorer d'avantage ce sous-ensemble.

Il existe plusieurs stratégies de parcours, parmi elles, la *stratégie de recherche en profondeur* : on choisit pour prochain noeud actif l'un des fils du noeud qui vient d'être divisé. Si aucun de ces noeuds n'est actif on revient en arrière (backtrack) dans l'arbre. Cette stratégie a plusieurs avantages :

- le nombre de noeuds actifs reste relativement faible et nécessite donc moins de mémoire ;
- l'évaluation d'un fils peut parfois profiter de l'évaluation du pére ;

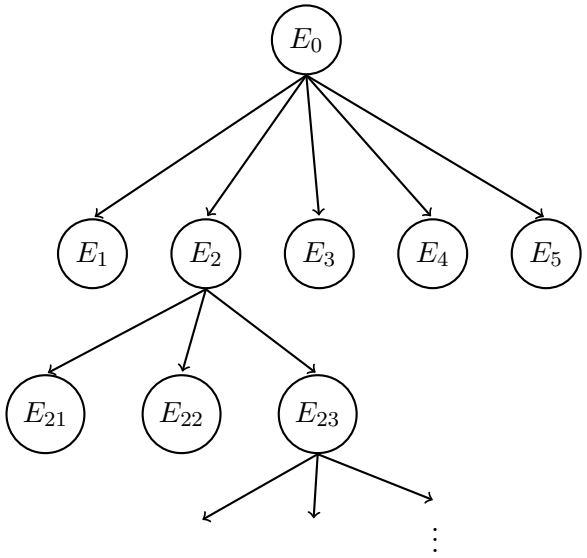

FIGURE 1.5 – Exemple d'arbre de recherche dans une procédure de séparation et d'évaluation

- il arrive que certains noeuds de l'arbre n'aient qu'un fils. Dans ce cas, il est inutile d'évaluer le noeud, mieux vaut évaluer son fils ;
- plus le noeud est situé en profondeur dans l'arbre, plus la probabilité de trouver une solution complète augmente, et donc la probabilité d'obtenir une valeur BS qui permet d'élaguer l'arbre de recherche augmente aussi.

Un pseudo code de l'algorithme d'une PSE avec exploration en profondeur est représenté dans la figure 1.6.

Input: Données du problème de minimisation
Output: α La variable à minimiser

```
1  //Initialisation
2  S_0 ← Empiler un premier noeud dans la pile PileNoeuds
3  α⁻ ← La borne inférieure de α
4  α⁺ ← La borne supérieure de α
5  //Boucle
6  while PileNoeuds ≠ VIDE and α⁻ ≠ α⁺ do
7      S ← Le neoud au sommet de PileNeouds
8      Dépiler S
9      if S(α) < α⁺ then
10         if S est une solution complète then
11             α⁺ ← S(α)
12         else
13             Créer les noeuds fils de S
14             Empiler les noeuds fils de S
15 return α⁺
```

FIGURE 1.6 – Algorithme d'une PSE avec exploration en profondeur

Chapitre 2

Etat de l'art des problèmes d'ordonnancement cyclique

Une présentation des différents problèmes de l'ordonnancement cyclique est faite dans ce chapitre. Ces problèmes sont présents dans plusieurs domaines (voir [KL97], [Pin05], [HP89], [HM95c]) car leur résolution consiste à organiser des opérations au fil du temps. Ces opérations peuvent être des tâches informatiques, des étapes d'un projet de construction, des opérations dans un processus de production, etc. Les contraintes dans un problème d'ordonnancement, en plus de l'ordre de priorité entre les opérations, sont dues à l'attribution de ressources limitées. Ces ressources peuvent être des processeurs, des employés, des machines. L'aspect cyclique de ces problèmes est relatif au fait que les opérations sont exécutées plusieurs fois (en fait une infinité de fois).

Dans un problème d'ordonnancement classique, un ensemble de tâches est exécuté une seule fois, l'ordonnancement établi optimise les fonctions objectifs telles que la minimisation du makespan, la minimisation des pénalités d'avance et de retard, etc (voir [RV10]). En revanche, dans un problème d'ordonnancement cyclique, un ensemble de tâches dites génériques est exécuté une infinité de fois, l'objectif étant généralement de minimiser le temps entre deux occurrences d'une même tâche, une occurrence d'une tâche étant une réalisation de celle-ci. Dans la section 2.1 nous présentons les principaux problèmes d'ordonnancement cyclique rencontrés dans la littérature et quelques méthodes de résolution classiques. Cet état de l'art s'inspire des états de l'art récent effectués dans l'article de Levner *et al* [LKdC10], la thèse de Maria Alejandra Ayala [Aya11] et l'habilitation à diriger des recherches de Rémy Dupas [Dup04a]. La section 2.2 se concentre sur le problème du Job-Shop cyclique, qui fait l'object de cette thèse.

2.1 Les différents problèmes d'ordonnancement cyclique

2.1.1 Problème d'ordonnancement cyclique de base

Le problème d'ordonnancement cyclique de base (GBCSP : General Basic Cyclic Scheduling Problem) est caractérisé par un ensemble de tâches élémentaires (ou génériques) $\mathcal{T} = \{1, ..., n\}$ qui seront répétées une infinité de fois. Chaque tâche i a une durée p_i, on

note $< i, k >$ la $k^{ème}$ occurrence de i telle que $k \in \mathbb{N}$. Dans un GBCSP, les tâches sont liées par des contraintes de précédence dites *uniformes*. Résoudre un GBCSP revient à trouver un ordonnancement cyclique σ qui détermine pour chaque occurrence $< i, k >$, sa date de début $t(i, k)$ tout en minimisant le temps de cycle asymptotique α.

Définition 2.1. Le temps de cycle asymptotique d'un ordonnancement w est défini par

$$\alpha = \lim_{k \to \infty} \frac{max_{i \in \mathcal{T}}(t(i, k) + p_i)}{k} \tag{2.1}$$

La minimisation du temps de cycle asymptotique α revient à maximiser le taux de production.

Définition 2.2. Le taux de production τ est défini comme suit

$$\tau = \frac{1}{\alpha} \tag{2.2}$$

Les dates de début de toutes les occurrences d'une tâche i sont liées comme suit :

$$\forall i \in \mathcal{T}, \forall k \in \mathbb{N} : \quad t(i, k) = t(i, 0) + \alpha k. \tag{2.3}$$

Puisque les dates de début de toutes les occurrences d'une tâche i sont liées, Nous pouvons définir un ordonnancement cyclique selon la proposition suivante :

Proposition 2.3. *Soit $t_i = t(i, 0)$ la date de début de la première occurrence de la tâche i et $\alpha \in \mathbb{R}^+$. un ordonnancement cyclique w est totalement défini par :*

- *Un ensemble $S_w = \{t_i \in \mathbb{R}^+ \mid i \in \mathcal{T}\}$*
- *α le temps de cycle*

Sachant que l'occurrence $< i, k+1 >$ ne peut pas commencer avant la fin de l'occurrence $< i, k >$, on peut écrire la constrainte de non dépassement suivante :

$$\forall i \in \mathcal{T}, \forall k \in \mathbb{N} : \quad t(i, k+1) \geq t(i, k) + p_i. \tag{2.4}$$

On exprime les contraintes de précédence entre deux tâches i et j, illustrées dans la figure 2.1, comme suit :

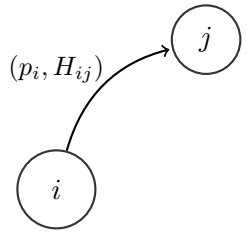

FIGURE 2.1 – Illustration de contraintes de précédence.

$$\forall i, j \in \mathcal{T}, \forall k \in \mathbb{N} : \quad t(i, k) + p_i \leq t(j, k + H_{ij}). \tag{2.5}$$

où $H_{ij} \in \mathbb{Z}$ est la hauteur qui peut aussi être considéré comme le décalage événementiel entre les occurrences des tâches i et j, cette valeur correspond à la différence entre les

occurrences de tâches dans un même cycle. Notons que dans la littérature, le terme p_i est souvent remplacé par une valeur L_{ij}, appelée longueur de la précédence. Dans cette thèse, dédiée au problème du Job-Shop cyclique, nous considérerons généralement le cas particulier où $L_{ij} = p_i$.

Le GBCSP peut alors s'écrire comme suit :

$$min\ \alpha \tag{2.6}$$

$$s.c. \quad t(i,k) + p_i \leq t(j, k + H_{ij}) \ \ \forall i, j \in \mathcal{T},\ k \in \mathbb{Z} \tag{2.7}$$

$$t(i,k) + p_i \leq t(i, k+1) \ \ \forall i \in \mathcal{T},\ k \in \mathbb{Z} \tag{2.8}$$

$$t(i,k) \geq 0 \ \ \forall i \in \mathcal{T} \tag{2.9}$$

$$t(i,k) \in \mathbb{R}^+ \ \ \forall i \in \mathcal{T},\ k \in \mathbb{Z}. \tag{2.10}$$

L'objectif est la minimisation du temps de cycle asymptotique α. Les contraintes (2.7) représentent l'ensemble des contraintes de précédence. Les contraintes (2.8) imposent le non dépassement entre les différentes occurrences d'une même tâche, ces contraintes de non dépassement sont incluses dans les contraintes de précédences si on considère que $H_{ii} = 1$ pour toute tâche i.

Il est connu (voir par exemple [HM95a]) que ce problème se résout aisément. Considérons un circuit du graphe. Définissons la longueur du circuit comme la somme des durées des tâches situées sur ce circuit et la hauteur du circuit comme la somme des hauteurs de ces mêmes tâches. Le temps de cycle minimum est égal au rapport le plus grand pour tous les circuits. Ce problème se résut en temps polynomial, par exemple par l'algorithme proposé dans [HM95a]. Nous reviendrons sur les approches de résolution du GBCSP dans la section 2.2 car elle est au coeur des méthodes de résolution de problèmes plus complexes, comme le Job-Shop cyclique.

Des contraintes de précédences linéaires, plus générales, sont introduites entre deux tâches génériques dans [HM09], ainsi qu'une caractérisation du comportement asymptotique d'un graphe linéaire. Les auteurs ont développé un algorithme pour calculer les fréquences optimales et évaluer la performance d'un graphe linéaire dans différentes situations. Ils ont également montré qu'un graphe linéaire peut être associé à un graphe uniforme avec le même comportement asymptotique.

2.1.2 Ordonnancement *K*-cyclique

Ce paragraphe est dédié à la distinction entre les ordonnancements 1-cycliques (ou 1-périodiques, ou encore strictement périodiques) et les ordonnancements K-cycliques (ou K-périodiques) avec $K > 1$ et $K \in \mathbb{N}$. Dans un ordonnancement 1-cyclique, tel que présenté dans la section précédente, la différence entre les dates de début de deux occurrences différentes d'une même tâche est égale au temps de cycle α, ainsi, dans un intervalle de temps de longueur α chaque tâche est traitée exactement une fois. En revanche, dans Dans un ordonnancement K-cyclique la différence entre la date de début de l'occurrence $< i, n >$ et la date de début de l'occurrence $< i, (n + K) >$ est égale au temps de cycle

α_K. Par conséquent, dans un intervalle de temps de longueur α_K chaque tâche est traité exactement K fois. Autrement dit, dans un ordonnancement K-cyclique, l'ordonnancement de K occurrences successives de chaque travail est fixe et se répète à un intervalle régulier de longueur α_K. L'équation 2.11 illustre les contraintes de précédence dans un ordonnancement K-cyclique entre les occurrences d'une tâche donnée.

$$\forall i \in \mathcal{T}, \forall l \in \mathbb{N} : \quad t(i, l + K) = t(i, l) + \alpha_K l. \tag{2.11}$$

Définition 2.4. Le temps de cycle α d'un ordonnancement K-cyclique est défini comme suit

$$\alpha = \frac{\alpha_K}{K} \tag{2.12}$$

Les ordonnancements 1-périodiques sont dominants pour les GBCSPs (voir [Han94]), mais ils ne le sont plus dans la plupart des cas dès qu'on introduit des contraintes de resources, en particulier dans les problèmes présentés dans cet état de l'art, dont le Job-Shop cyclique.

2.1.3 Problème d'ordonnancement cyclique à machines parallèles

Dans un GBCSP, on peut considérer que chaque tâche est excécutée sur sa propre machine dédiée, tandis que dans le cas d'ordonnancement cyclique à machines parallèles (PSIP : Periodic Scheduling on Identical Processors), le nombre de machines m est inférieur au nombre de tâches génériques. Les machines sont identiques et chaque tâche peut être traitée par n'importe quelle machine. L'ordonnancement s'effectue en deux étapes : la première étape consiste à affecter les tâches aux machines et la deuxième étape établit l'ordre de passage des tâches sur chaque machine (voir [Mun96]).

Résoudre un PSIP revient à trouver un ordonnancement cyclique w qui détermine pour chaque occurrence k de la tâche i, notée $< i, k >$, sa date de début $t(i, k) \geq 0$ et la machine $r(i, k)$ sur laquelle l'occurrence $< i, k >$ va être exécutée. L'ordonnancement w doit vérifier les contraintes de précédence et le fait qu'une machine n'exécute qu'une seule tâche à la fois. L'objectif étant toujours de minimiser le temps de cycle asymptotique α.

Dans un problème de machines parallèles un ensemble des opérations (tâches) $\{O_i\}_{1 \leq i \leq n}$ est exécuté sur m processeurs (machines) identiques. Pour chaque exécution d'une opération i, il est nécessaire d'utiliser un des m processeurs pendant p_i unités de temps. La préemption n'est pas autorisée. Les contraintes de ressources sont liées au nombre limité de processeurs et au nombre d'opérations qui sont exécutées à une période donnée. Un ensemble de contraintes de précédence est également à considerer. Un ordonnancement réalisable est un choix de dates de début $\{t_i\}_{1 \leq i \leq n}$, tel que les contraintes de ressources et de précédence soient respectées.

Le problème d'ordonnancement cyclique à machines parallèles a été abordé dans plusieurs travaux [Mun91, HM95b, HM95c, Mun96, Chr00]. Des cas polynomiaux ont été exhibés (par exemple lorsque le graphe de précédence est un circuit), des règles de dominance sur les allocations de ressources ont été définiées, des algorithmes de liste (gloutons)

tels qu'il existe une liste menant à la solution optimale ont été proposés. Des bornes (ou ratio de compétitivité) entre la valeur d'un algorithme de liste et la solution optimale du problème, qui étendent la fameuse borne de Graham pour le problème à machine parallèle ont également été établies. A titre d'exemple, dans [GS94], en notant α_{opt} le temps de cycle optimal, α_L le temps de cycle obtenu par un algorithme de liste donnant un ordonnancement strictement périodique et p_{max} la plus grande durée des tâches, les auteurs montrent que

$$\lambda_L \leq (2 - \frac{1}{m})\lambda_{opt} + \frac{m-1}{m}(p_{max} - 1).$$

Il faut finalement retenir que le PSIP est un problème NP-difficile au sens fort et que le cas particulier de PSIP à seulement deux machines est NP-difficile.

Une propriété fondamentale (voir par exemple [HM95a]) est en outre partagée par les problèmes d'ordonnancement cyclique sur machines parallèles et les problèmes d'ordonnancement avec contraintes de ressources qui seront présentés dans la suite de cet état de l'art, y-compris le problème du Job-Shop cyclique, objet de cette thèse. Les ordonnancement strictement périodiques ne sont plus dominant pour la minimisation du temps de cycle moyen, contrairement au cas du GBCSP. Pour minimiser ces temps de cycle il faut en toute rigueur considérer des ordonnancements K-périodiques.

Ainsi, Chrétienne [Chr00] obtient une borne pour les algorithmes de liste générant des ordonnancements dits réguliers (non nécessairement strictement périodiques). Le quotient de compétitivité est de $(2 - (\min\{H^*, m\})/m)$ où H^* est la hauteur minimale d'un circuit dans un graphe de précédence dit réduit.

Néanmoins, la simplicité de mise en oeuvre des ordonnancements strictement périodiques (tels que $K_i = 1$) incite à restreindre l'ensemble des solutions à cette classe d'ordonnancement.

2.1.4 Problème Job-Shop Cyclique

Le problème de Job-Shop cyclique (CJSP : Cyclic Job-Shop Scheduling Problem) est aussi un problème NP-difficile (voir [Han94]). Un exemple de CJSP est représenté dans la figure 2.2, cet exemple est composé de huit tâches génériques regroupées en trois travaux (contraintes uniformes) et qui utilisent quatre machines. Les travaux sont représentés par les différentes flèches qui vont d'un point "Début" à un point "Fin", par exemple, le travail représenté avec des flèches doubles est composé de trois tâches élémentaires : la première utilise la machine 1, la deuxième utilise la machine 2 et la troisième utilise la machine 3.

Comme dans un PSIP, le nombre de machines dans un CJSP est inférieur au nombre de tâches génériques, mais à la différence du PSIP, ces machines ne sont pas identiques et il n'y a pas de problème d'affectation : toute tâche est préaffectée à une des machines, qu'elle doit donc partager avec toutes les autres tâches également affectées à cette machine. Par conséquent, les machines jouent un rôle important et les contraintes qu'elles génèrent sont ajoutées au CJSP. L'objectif de la résolution d'un CJSP est de trouver un ordonnancement périodique ayant un temps de cycle asymptotique minimal et vérifiant toutes les contraintes de précédence et de ressource.

Nous définisssons simplement dans cette section le CJSP comme un GBCSP auquel

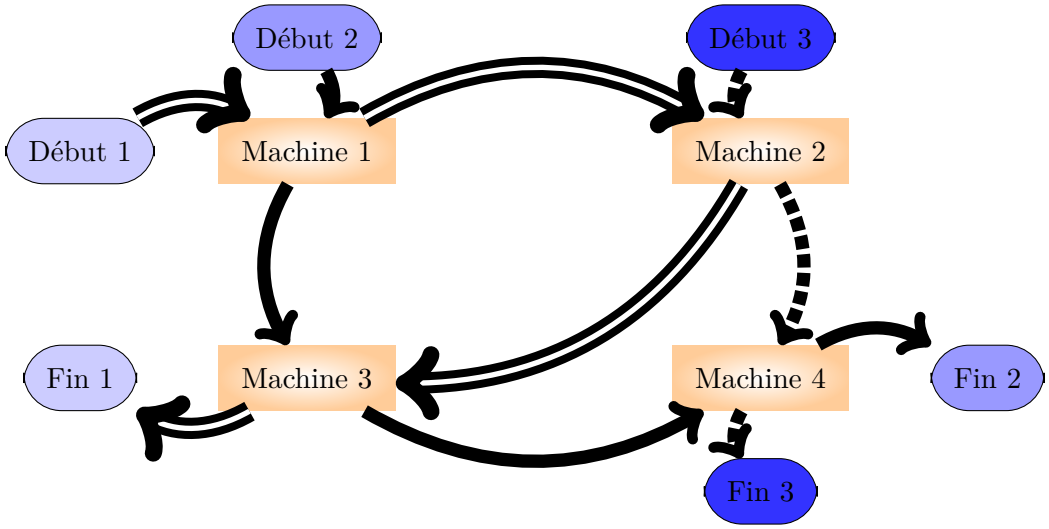

FIGURE 2.2 – Représentation de jobshop.

on ajoute des contraintes de ressources. Soit $\mathcal{M} = \{1, ..., m\}$ un ensemble de machines où $m < n$. La tâche i est exécutée sur la machine $M(i) \in \mathcal{M}$ sans interruption pendant toute sa durée p_i. Le chevauchement de deux occurrences de tâches différentes utilisant la même machine représente un conflit de ressources. Ce conflit est évité en ajoutant des contraintes disjonctives. Les contraintes de ressources dites disjonctives peuvent s'écrire comme suit : $\forall i, j \in \mathcal{T},\ \forall k, l \in \mathbb{N}$ tels que $M(i) = M(j),\ i \neq j\ and\ k \neq l$:

$$t(i,k) \leq t(j,l) \Rightarrow t(i,k) + p_i \leq t(j,l). \tag{2.13}$$

Les contraintes de précédence possèdent en fait dans le problème de Job-Shop classique une structure particulière liée à l'existence de n/m travaux comportant chacun m tâches, chacune étant allouée à une machine différente. Les tâches d'un travail sont liées par des contraintes de précédence de type «chaîne» représentant l'ordre de passage du travail sur les machines. Par référence au problème d'ordonnancement cyclique de base, entre deux tâches consécutives d'une chaine une contrainte de précédence de hauteur nulle est présente. Nous reviendrons sur la structure de précédence du Job-Shop cyclique dans la section 2.2.

Exemple 2.5. Un ensemble de quatre tâches élémentaires regroupées en deux travaux doit être exécuté de façon cyclique sur deux machines. Le tableau 2.1 indique le détail de ce CJSP et la figure 2.3 le représente.

TABLE 2.1 – Données de l'exemple de CJSP

Travail	1		2	
Tâche	t_1	t_2	t_3	t_4
Durée	5	4	2	3
Machine	1	2	1	2

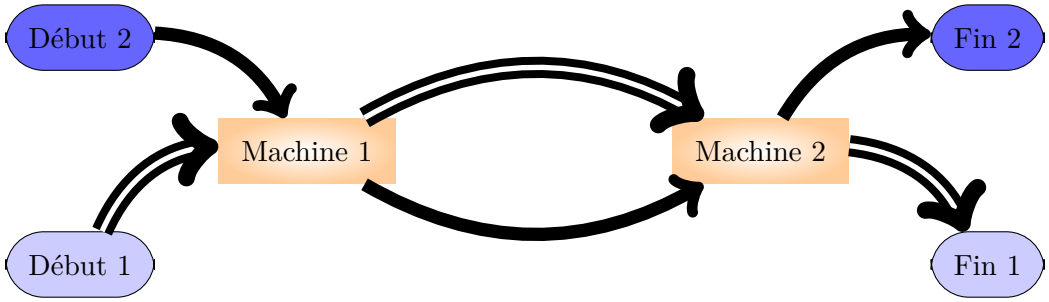

FIGURE 2.3 – Représentation de l'exemple de CJSP.

La figure 2.4 représente une solution du CJSP de cet exemple, avec un temps de cycle égal à 12, où le travail 1 est effectué avant le travail 2. Une deuxième solution est représentée dans la figure 2.5 et qui correspond à un meilleur temps de cycle, 11, dans cette deuxième solution le travail 2 est effectué avant le travail 1.

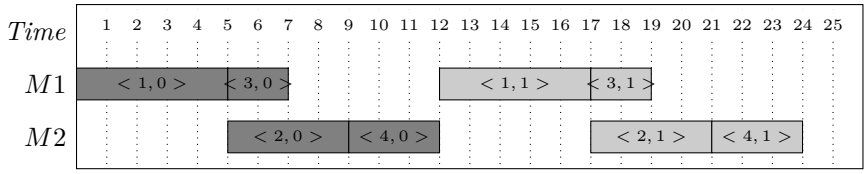

FIGURE 2.4 – Représentation d'une solution de l'exemple de CJSP.

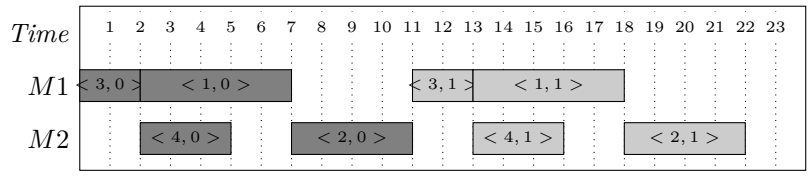

FIGURE 2.5 – Représentation d'une deuxième solution de l'exemple de CJSP.

Le Work-In-Process (WIP), introduit dans la partie 2.2.2 et expliqué plus en détail dans la partie 3.1.1, correspond au le nombre maximal de travaux en cours. En considérant cette notion, les deux solutions précédentes sont des ordonnancements 1-cycliques avec un WIP égal à 1. Si on augmente le WIP, on peut obtenir de meilleures solutions en terme de minimisation du temps de cycle du CJSP. En effet, la figure 2.6 représente un ordonnancement 1-cyclique avec un WIP égal à 2 dont le temps de cycle est égal à 7, avec le même ordre de passage des travaux, la solution de la figure 2.5 qui est 1-cyclique avec un WIP de 1 affiche un plus grand temps de cycle : 11.

Dans [Han90], l'auteur montre que les solutions K-périodique sont dominantes pour la minimisation de temps de cycle dans les CJSPs.

Exemple 2.6. La figure 2.7 représente une solution 2-cyclique pour le CJSP de l'exemple 2.5. Dans cette solution, la différence entre le 1-cyclique et le 2-cyclique est perceptible. En effet, la deuxième occurrence du travail 2 est considérée comme un nouveau travail. Ainsi, il n'y a plus de contrainte de précédence entre la première occurence $< 4, 0 >$ de la

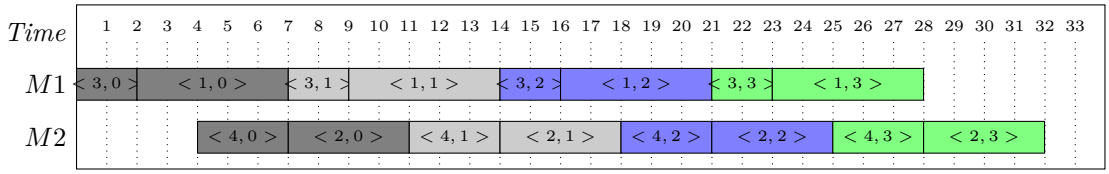

FIGURE 2.6 – Représentation d'une solution de l'exemple de CJSP 1-cyclique avec WIP=2.

tâche t_4 et la deuxième occurence $< 3, 1 >$ de la tâche t_3 qui peuvent s'executer en même temps.

Cette solution est un exemple de la dominance des ordonnancements 2-cyclique par rapport aux ordonnancements 1-cyclique. En effet, le temps de cycle de cette solution est également de 7.

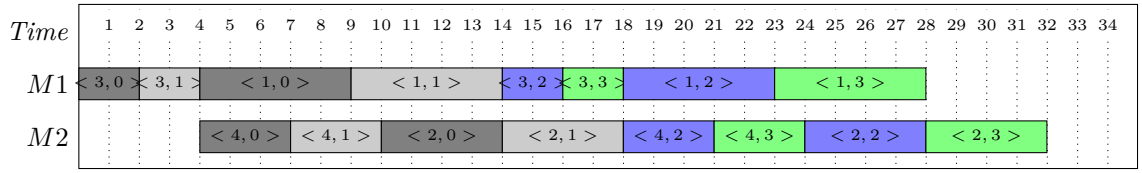

FIGURE 2.7 – Représentation d'une solution de l'exemple de CJSP 2-cyclique.

Pour plus de détails sur le Job-Shop cyclique, le lecteur peut se référer à [Han94, BCL02, BK05, Rou92, SL02, Kam06]. Nous présenterons les différentes approches proposées pour sa résolution dans la section 2.2.

2.1.5 Autres problèmes d'ordonnancement cyclique

Un certain nombre de travaux concernant l'ordonnancement cyclique dans des environnements différents peuvent être rencontrés dans la littérature. Nous reprenons ici brièvement quelques résultats que l'on pourra consulter en détails dans les états de l'art [Dup04a, LKdC10, Aya11].

Ordonnancement cyclique dans les cellules robotisées, hoist scheduling et production flexible manufacturière

Une cellule robotisée peut être définie schématiquement comme un atelier de production (par exemple de type Job-Shop) dans lequel un ou plusieurs robots interviennent pour transporter des pièces d'une machine à l'autre. Un cas typique est l'atelier de type Flow-Shop robotisé où les travaux passent sur les machines dans le même ordre. Lorsqu'une tâche est terminée sur une machine, alors la machine est bloquée jusqu'à ce qu'un robot viennent enlever la pièce et la déplacer sur la machine suivante. Il s'agit alors de déterminer l'ordre d'entrée des travaux sur la première machine, puis d'ordonnancer les mouvements des robots de manière à optimiser une fonction objectif, qui peut être le makespan d'une itération ou bien comme considéré dans cette thèse, le temps de cycle moyen. Trois niveaux de décisions apparaissent : (1) déterminer la séquence d'entrée (2) ordonner les mouvements des robots (2) déterminer les dates de début des tâches. La spécificité de ce

problème par rapport au problème d'ordonnancement cyclique considéré dans la présente thèse provient donc de la détermination du séquencement des mouvement du robot.

Notons qu'un cas d'atelier robotisé, connu sous le nom de « hoist scheduling problem » a été particulièrement étudié dans la littérature [LW89, CC05, CCP95, PU76, Dup04a, BMBV12] sur la base de problèmes issus notamment de la galvanoplastie où des pièces doivent subir des traitements de surface dans des bains chimiques ou électrolytiques. En plus des mouvements des robots, des problèmes d'attente limitéee entre différents bains viennent compliquer le problème.

Plus généralement, les systèmes flexibles de production manufacturière (SFPM) qui ont pour caractéristique une forte automatisation alliée à une flexibilité concernant les reconfiguration de l'atelier pour opérer différents types de produits sont une source inépuisable de problèmes complexes de problèmes d'ordonnancement cyclique robotisés [Dup04a]. Plus de détails sur des exemples de problèmes issus des SFPM et les méthodes de résolution associées peuvent être trouvés dans [ACG92, Ken99, OCG97, KG00, XHG02, KCG02, HKDG08].

A part pour des versions simplistes, ces problèmes sont généralement NP-difficiles. Plus de détails sur l'ordonnancement dans les cellules robotisées peuvent ête consultés dans [CdK97, KL97, CKvdKL00, Bra08, DGSS05, CC09, MT02, BBG12].

Problèmes d'ordonnancement cyclique sous contraintes de ressources

Au delà des problèmes d'atelier, où les ressources sont des machines ou des robots de type "disjonctif", c'est à dire qu'une ressource ne peut exécuter qu'une seule tâche à un instant donné, des travaux récents considèrent des problèmes impliquant des ressources plus complexes, appelées ressources cumulatives ou discrètes selon les auteurs. Ces problèmes apparaissent par exemple dans le contexte de l'ordonnancement d'instruction au sein des compilateurs pour les architectures parallèles [ED97, Dup04b]. Une ressource cumulative est définie par une capacité maximale instantanée (par exemple une taille mémoire) et chaque tâche demande éventuellement tout au long de son exécution un nombre limité mais possiblement supérier à 1 d'unités de cette ressource. Ainsi à tout moment le nombre de tâches utilisant cette ressource exécutées en parallèle est limité par le fait que la somme des demandes des tâches ne doit pas excéder la capacité de la ressource. En ordonnancement acyclique, un problème très classique impliquant ce type de ressource est le RCPSP (Resource-Constrained Project Scheduling Problem). Il faut noter que lorsqu'une seule resource est disponible et que les tâches demandent chacune une seule unité, on obtient le problème à machines parallèles. Aussi, les méthodes d'ordonnancement cyclique dans ce contexte sont souvent adaptées des méthodes utilisées pour résoudre le RCPSP d'un côté et des techniques de résolution du PSIP d'un autre côté. Plus de détails sur de tels problèmes peuvent être trouvés dans [ED97, Dup04b, DAA08, Dup07, BH11, Aya11, ABAH13].

2.2 Les différentes approches de modélisation et de résolution de CJSP

Plusieurs points de vue ont été utilisés pour aborder cette classe de problèmes. La plupart d'entre eux tire parti de la théorie des graphes, de la programmation linéaire en nombres entiers, des réseaux de Petri et de l'algèbre $(max, +)$.

2.2.1 Graphes d'événements temporisés (Réseaux de Petri)

La modélisation d'un CJSP par un GET se fait usuellement en trois étapes [RH08] :

- *Modélisation du processus* pour chaque travail : pour celà, chaque tâche est représentée par une transition t. Les différentes transitions associées aux tâches appartenant à un travail donné, sont séparées les unes des autres par des places (symbolisant les stocks ou plus généralement les moyens de stockage). A chaque transition est associé un temps de franchissement qui est égal à la durée de la tâche représentée par cette transition.
- *Modélisation de la cyclicité du système* : en supposant que les travaux ne s'entrelassent pas, une place supplémentaire vient augmenter le modèle créé à la première étape. En supposant aussi que lorsqu'une occurrence d'un travail donné est terminée, l'occurrence suivante de ce même travail peut commencer immédiatement, cette hypothèse est symbolisée par un rebouclage sur la place supplémentaire des transitions d'entrée et de sortie du modèle. Les circuits ainsi créés sont appelés des *circuits de fabrication*. La figure 2.8 représente les circuits de fabrication du CJSP de l'exemple 2.5.

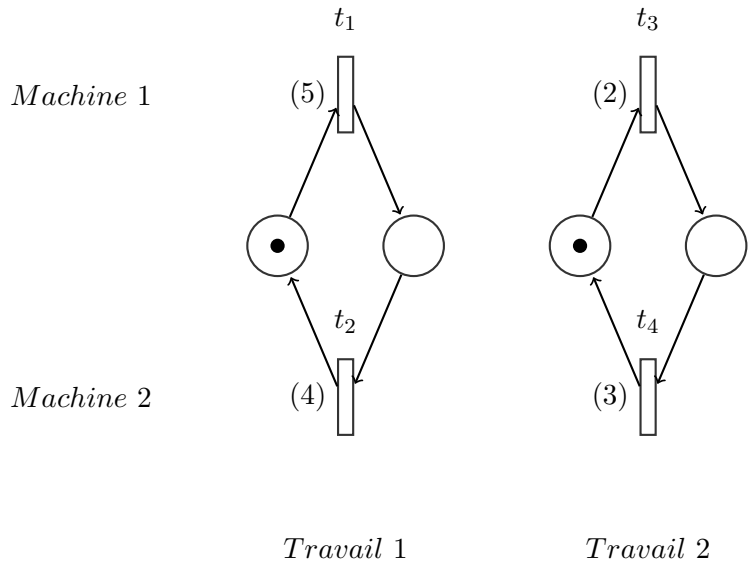

FIGURE 2.8 – Circuits de fabrication du CJSP de l'exemple 2.5

- *Modélisation des contraintes disjonctives* : les transitions représentant les tâches exécutées sur une machine donnée sont intégrées dans un circuit élémentaire appelé *circuit de commande*. L'ordre de parcours du circuit de commande ainsi créé est

l'ordre de passage des travaux dans la machine. La figure 2.9 représente le RdP qui modélise le CJSP de l'exemple 2.5 où les places des circuits de fabrication sont désignées par p et les places des circuits de commande sont désignées par c.

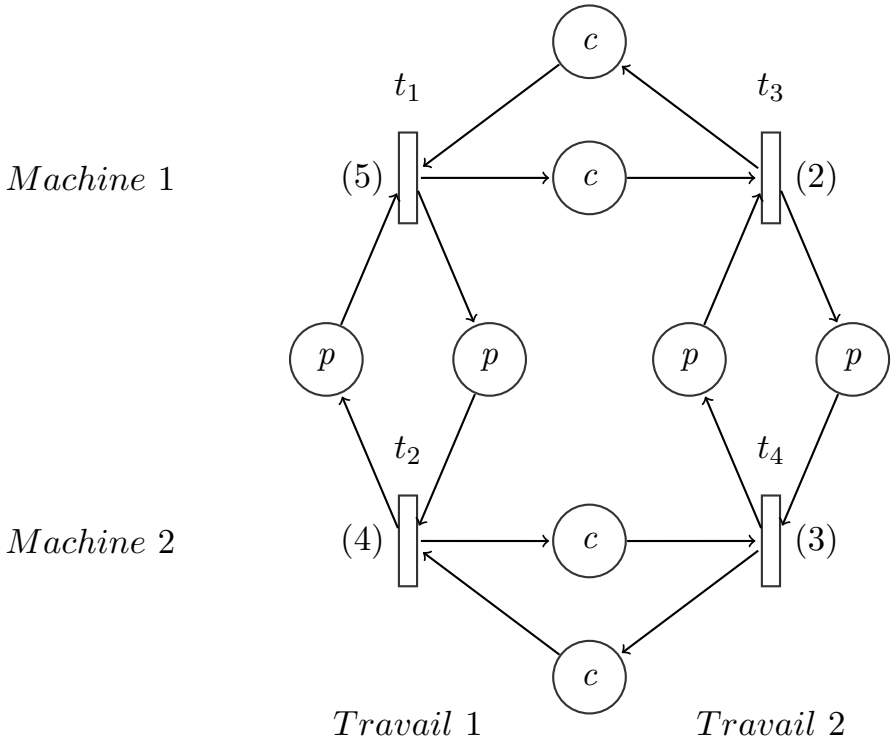

FIGURE 2.9 – Réseau de Petri modélisant le CJSP de l'exemple 2.5.

Cette modélisation du CJSP par des RdP consitue en général une première étape [Coh95], pour évaluer les performences du système modélisé et résoudre le problème d'optimisation, l'étape suivante est la modélisation linéaire des GETs selon l'un des deux points de vue cités dans la partie 1.4.2.

2.2.2 Théorie des graphes

Un graphe orienté $G = (T, E)$ avec T un ensemble de sommets et E un ensemble d'arcs peut être associé à un GBCSP tel que : un sommet (resp. un arc) de G correspond à une tâche élémentaire (resp. une contrainte uniforme) dans le GBCSP. Chaque arc (i, j) de G est doublement valué : la première valeur est la longueur $L_{ij} \in \mathbb{R}$ et la deuxième valeur est la hauteur $H_{ij} \in \mathbb{Z}$. Les contraintes uniformes exprimées dans l'équation (2.5) s'écrivent aussi comme suit :

$$\forall (i,j) \in E, \forall k \in \mathbb{N} : \quad t(i,k) + L_{ij} \leq t(j, k + H_{ij}). \tag{2.14}$$

Comme déjà évoqué dans la section 2.1, un GBCSP est dit consistant si, et seulement si, il admet une solution réalisable. Le graphe G permet de vérifier la consistance du GBCSP en utilisant le théorème suivant (Voir [Han94], [CC88], [CDQV85]) :

Théorème 2.7. *Un GBCSP est consistant si et seulement si chaque circuit du graphe*

associé G a une hauteur positive.

Par ailleurs, le temps de cycle minimal d'un GBCSP est donné par le circuit critique du graphe associé. Plusieurs algorithmes existent pour trouver le circuit critique d'un graphe orienté doublement valué. Nous pouvons d'abord citer l'algorithme de Gondran et Minoux [GM95] qui a une complexité en $O(n^3 log(n))$. Un deuxième algorithme est celui de Karp de complexité $O(n^3)$ (voir [Kar78, DG98]), il ne fonctionne que lorsque toutes les hauteurs H_{ij} qui existent sont égales à 1. Quand $H_{ij} \geq 0$, quelques modifications sont nécessaires (voir [BCOQ92]). Quant au cas où $H_{ij} \leq 0$ fréquemment rencontré dans les GBCSP (*cf.* l'exemple de base de [Han94]), une première tentative pour les prendre en compte dans l'algorithme de Karp a été faite dans [Hou11]. Le troisième algorithme est celui de Howard (voir [How60]), initialement conçu pour les processus décisionnels de Markov, il a été adapté à l'algèbre $(max, +)$ dans [CTCG$^+$98]. Bien que la complexité de l'algorithme de Howard reste non prouvée, il montre d'excellentes performances et un temps de calcul presque linéaire.

Dans le cas d'un CJSP, le graphe orienté $G = (T, E)$ associé est tel que : un sommet (resp. un arc) de G correspond toujours à une tâche élémentaire (resp. contrainte). Tout arc uniforme (i, j) de G possède deux valeurs $L_{ij} = p_i$ et H_{ij}. Tout arc disjonctif a aussi deux valeurs : la durée $L_{ij} = p_i$ et le décalage événementiel $K_{ij} \in \mathbb{Z}$. Pour toute machine s, si $i \in T_s$ et $j \in T_s$ alors : $K_{ij} + K_{ji} = 1$ (voir [Han94] pour plus de détails). Deux sommets factices sont ajoutés au graphe, ils représentent le début et la fin d'une occurrence k. L'arc qui relie ces deux sommets est également doublement valué avec une longueur nulle et une hauteur H_{es} non négative, appelée Work-In-Process (WIP). Cette hauteur représente le nombre maximal de travaux en cours, plus d'explications sur cette valeur de la hauteur sont disponibles dans [BK08].

La figure 2.10 représente le graphe associé au CJSP de l'exemple 2.5. En plus des arcs disjonctifs, on retrouve la structure en chaîne des contraintes de précédence entre les tâches d'un même travail (arcs orientés (i, j)) avec une longueur égale à la durée de i et une hauteur nulle présentée dans la section 2.1.

2.2.3 Algèbre (max,+)

Le CJSP peut être considéré, s'il est consistant, comme un système dynamique à événements discrets qui peut être modélisé à l'aide de fonctions dateurs (voir 1.4.2). La première étude concernant les problèmes d'ordonnancement cyclique mettant en jeu l'algèbre $(max, +)$ a été établie par [CMQV89]. Dans ce papier, les auteurs modélisent une solution d'ordonnancement par un système $(max, +)$-linéaire autonome :

$$x(k) = \bigoplus_{i \in \mathbb{Z}} A_i x(k - i), \tag{2.15}$$

où $x(k)$ est un vecteur contenant les $k + 1$-ème dates d'execution des tâches et A est la matrice d'évolution.

Dans le cas d'un système causal, *i.e.* $i \geq 0$, il est possible d'obtenir une formulation plus compacte. Celle-ci consiste en une extension du vecteur d'état et la résolution de

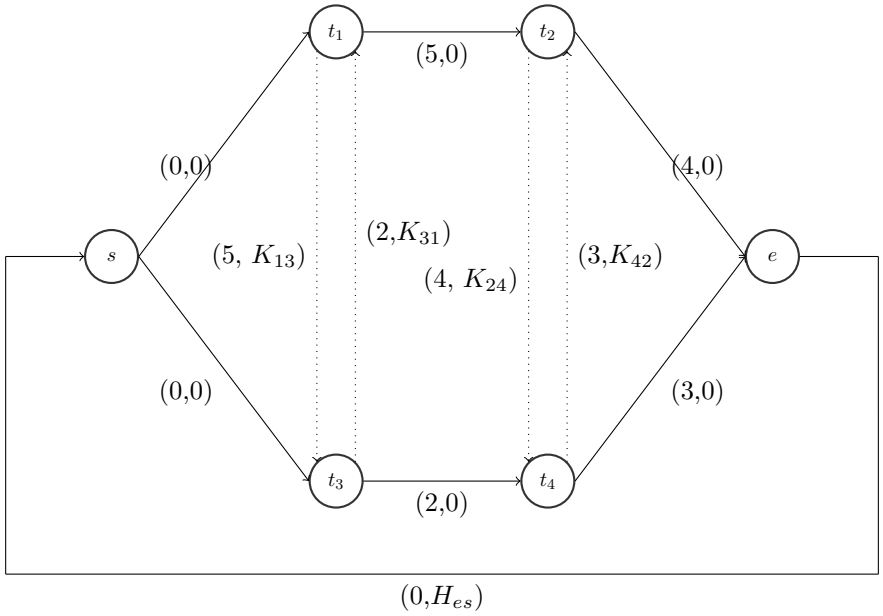

FIGURE 2.10 – Graphe associé de l'exemple de CJSP

l'équation implicite (plus de détails dans [BCOQ92]). On obtient alors

$$x(k) = Ax(k-1). \tag{2.16}$$

avec A une matrice irréductible.

Ces systèmes ont été étudiés dans le chapitre 1.4.2.

Dans le cas, où le système n'est pas causal, les résultats sont moins évidents. Une première tentative a été faite dans [Hou11] mais la manipulation de systèmes non causaux rend l'évaluation de performance difficile. Concernant les ordonnancements cycliques, un Work-In-Process supérieur à un induit généralement un système non causal. Le diagramme de Gantt d'un tel ordonnancement est commposé de motifs identiques qui s'entrelacent comme dans la figure 2.6.

L'apport de la théorie des systèmes $(max, +)$ est donc limité pour ce type de problème. Néanmoins, les travaux de [CTCG$^+$98] ont montré qu'il était possible de calculer le temps de cycle de tels systèmes à l'aide de l'algorithme de Howard. En effet, les auteurs adaptent cet algorithme afin de répondre à la question de l'évaluation du temps de cycle pour les systèmes définis par (2.15).

2.2.4 Programmation linéaire mixte

Un CJSP peut être transformé en un programme d'optimisation mathématique comme suit :

$$min\ \alpha \tag{2.17}$$

$$s.c. \quad t_j - t_i \geq p_i - \alpha \times H_{ij} \tag{2.18}$$

$$t_j - t_i \geq p_i - \alpha \times K_{ij} \tag{2.19}$$

$$K_{ij} + K_{ji} = 1. \tag{2.20}$$

Où t_i est la date de début de l'occurrence $< i, 0 >$

Quelques modifications sont nécessaires car les contraintes de ressources engendrent une contrainte non-linéaire avec $\alpha \times K_{ij}$. Ainsi, le débit $\tau = \alpha^{-1}$ est introduit avec une nouvelle variable $u_i = t_i \times \alpha^{-1}$ (Voir [Han94], [BK08]).

Le problème de minimisation du temps de cycle asymptotique α devient donc un problème de maximisation du débit τ. L'équation (2.14) qui représente les contraintes uniformes, s'écrit donc comme suit :

$$u_j - u_i \geq \tau \times L_{ij} - H_{ij} \quad \forall i, j \in \mathcal{T} \tag{2.21}$$

Les contraintes disjonctives sont exprimées à travers les trois équations suivantes :

$$u_j - u_i \geq \tau \times p_i - K_{ij} \quad \forall s \in \mathcal{M},\ \forall i, j \in T_s \tag{2.22}$$

$$K_{ij} + K_{ji} = 1 \quad \forall i, j \in \mathcal{T}. \tag{2.23}$$

$$K_{ij} \in \mathbb{Z}. \tag{2.24}$$

De plus :

$$\forall i \in \mathcal{T},\ u_i \geq 0\ et\ \tau \geq 0. \tag{2.25}$$

Ce qui conduit au programme linéaire mixte suivant :

$$max\ \tau \tag{2.26}$$

$$s.c. \quad u_j - u_i \geq \tau \times p_i - H_{ij} \tag{2.27}$$

$$u_j - u_i \geq \tau \times p_i - K_{ij} \tag{2.28}$$

$$K_{ij} + K_{ji} = 1. \tag{2.29}$$

2.2.5 Théorie des tas

Une approche originale est apparue pour considérer le problème d'ordonnancement cyclique, elle a été développée dans [GM99]. Cette approche est abordée en détails dans le chapitre 4.1.

2.2.6 Résolution

Contrairement à son équivalent acyclique, on trouve peu de méthode dans la littérature pour la résolution du CJSP. Cela est d'autant plus vrai pour les méthodes exactes. Dans cette thèse nous comparerons les méthodes que nous proposons avec la procédure de séparation et d'évaluation proposée par [Han94] et basée sur le formalisme des graphes introduits ci-dessus. Brucker et Kampmeyer résolvent dans [BK08] de petites instances au moyen du MLP présenté plus haut. Notons que les programmes linéaires en nombres entiers proposés pour le RCPSP cyclique dans [Dup04b, ED97] peuvent également s'appliquer au Job-Shop cyclique. Il s'agit de modèles à variables binaires x_{it} indicées par le temps où $x_{it} = 1$ signifie que la tâche générique démarre à l'instant t. L'inconvénient de ces méthodes est une très grande sensibilité soit à l'horizon de temps pour le PLNE proposé dans [Dup04b], soit à la période α pour le PLNE proposé dans [ED97] avec un écroulement des performances pour des grandes valeurs de ces paramètres. Plus récemment une méthode de programmation par contraintes a été proposée dans [BLBM11] pour le RCPSP cyclique et peut donc s'appliquer au Job-Shop cyclique. Bien que cette méthode obtienne de bons résultats, elle a toutefois du mal à prouver l'optimalité sur les instances testées.

Hormis ces rares méthodes exactes, des approches heuristiques ont été proposées comme par exemple la méthode tabou de [BK05] ou l'algorithme génétique de [CDG05]. Nous nous focalisons néanmoins dans cette thèse sur les approches exactes et nous renvoyons le lecteur à ces articles pour plus de détails.

Chapitre 3

Approche par la théorie des graphes

Ce chapitre est dédié à la présentation de méthodes de résolution exacte des CJSP. La première est une PSE basée sur les graphes, développée avec deux variantes, la différence entre ces deux versions est que l'une permet de mieux élager l'arbre de recherche mais demande plus de calcul à chaque noeud par rapport à l'autre version. La deuxième procédure est également une PSE basée sur les graphes proposée dans [Han94], développée en deux versions aussi, la différence étant uniquement l'algorithme de calcul des temps de cycle (évaluation de solutions) et ce pour avoir une meilleure comparaison entre les procédures. Un aperçu de ce travail est également disponible dans [FBH12, BFH12].

3.1 Nouvelle procédure de séparation et d'évaluation pour le CJSP

Nous proposons une procédure de branch and bound pour résoudre les CJSP, elle repose sur deux piliers principaux : la consistance d'un graphe pour obtenir des bornes sur les décalages événementiels et l'algorithme de Howard pour calculer les temps de cycle et évaluer un noeud. Le paragraphe suivant établit d'abord les concepts théoriques sous-jacents avant de décrire la structure de l'algorithme et d'illustrer cet algorithme par l'utilisation de l'exemple 2.5.

3.1.1 Concepts théoriques

Dans ce qui suit, des bornes inférieures et des bornes supérieures de certaines valeurs sont définies, en utilisant la modélisation par les graphes d'un CJSP. Ces valeurs sont : le temps de cycle, les décalages événementiels, les hauteurs, les amplitudes, etc. Les bornes supérieures (resp. bornes inférieures) sont représentées par un exposant $+$ (resp. $-$) ajouté à la notation habituelle de la valeur concernée.

Définition 3.1. La hauteur d'un circuit est la somme des hauteurs (ou décalages événementiels) de tous les arcs de ce circuit.

Définition 3.2. Un graphe G est dit consistent si et seulement si tous les circuits C de G ont une hauteur plus grande ou égale à 1.

Ajouter un nouvel arc disjonctif à un graphe crée au moins un circuit et ne compromet pas la consistance du graphe si la solution reste réalisable. Par conséquent, chaque décalage événementiel K_{ij} d'un arc disjonctif (i, j) a une borne inférieure évidente qui garanti la consistance du graphe. D'où le théorème suivant :

Théorème 3.3. *On considère un CJSP où les tâches i et j sont exécutées sur la même machine. On peut déduire une borne inférieure notée K_{ij}^- pour le décalage événementiel K_{ij} telle que :*

$$K_{ij}^- = 1 - \min\{H(\mu) \mid \mu \textit{ chemin de } j \textit{ à } i \textit{ dans } G\}. \tag{3.1}$$

Preuve : Une solution réalisable nécessite une hauteur au moins égale à 1 pour tous circuit du graphe c.à.d :

$$\forall C \in G : H(C) \geq 1.$$

Donc, pour tout arc disjonctif (i, j), on obtient :

$$K_{ij} + \{H(\mu) \mid \mu \text{ chemin de } j \text{ à } i \text{ dans } G\} \geq 1.$$

Donc

$$K_{ij} \geq 1 - \{H(\mu) \mid \mu \text{ chemin de } j \text{ à } i \text{ dans } G\}.$$

Cette inégalité est vraie pour tous les chemins de j à i dans G, on prend donc le plus court chemin. Ainsi,

$$\forall i, j \in \mathcal{T}, \mid M(i) = M(j), K_{ij} \geq K_{ij}^-$$

avec :

$$K_{ij}^- = 1 - \min\{H(\mu) \mid \mu \text{ chemin de } j \text{ à } i \text{ dans } G\}.$$

□

Corollaire 3.4. *Puisque $K_{ij} + K_{ji} = 1$ (voir [Han94]), nous pouvons déduire un intervalle pour chaque variable K_{ij} comme suite :*

$$K_{ij}^- \leq K_{ij} \leq 1 - K_{ji}^- \tag{3.2}$$

Preuve : Nous avons $K_{ji}^- \leq K_{ji}$ et $\forall i, j \in T,\ K_{ij} \in \mathbb{Z}$ alors, $K_{ij} + K_{ji}^- \leq K_{ji} + K_{ji}$. Puisque $K_{ij} + K_{ji} = 1$ alors, $K_{ij} \leq 1 - K_{ji}^-$.

□

Exemple 3.5. Ce concept se vérifie facilement à travers l'exemple 2.5. Les tâches t_1 et t_3 sont exécutées sur la même machine, deux arcs disjonctifs (un de t_1 à t_3 et l'autre de t_3 à t_1) sont donc nécessaires pour déterminer une priorité entre les deux tâches. L'arc

disjonctif qui part de t_1 à t_3 induit un nouveau cycle $C = (t_1, t_3, t_4, e, s, t_1)$, représenté dans la figure 3.1. La hauteur de ce cycle $H(C) = K_{13} + 0 + 0 + 2 + 0$ doit être ≥ 1 pour que la solution reste réalisable. Ce qui donne la borne inférieure de $K_{13} \geq -1$.

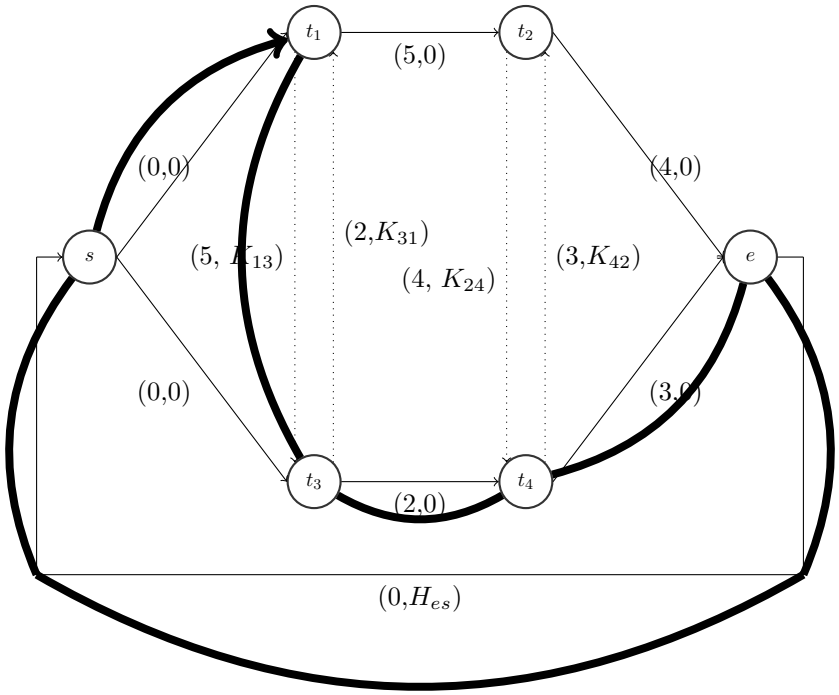

FIGURE 3.1 – Graphe CJSP avec le circuit cr*éé* par l'arc disjonctif $(2, 4)$

Dans ce CJSP, $K_{13}^- \leq K_{13} \leq 1 - K_{31}^-$. D'où : $-1 \leq K_{13} \leq 2$.

Deux bornes inférieure pour le temps de cycle peuvent être obtenues : la première à partir du CJSP sans les contraintes de ressources (ou du GBCSP associé), la deuxième est donnée par les limites des machines.

Théorème 3.6. *Le GBCSP fournit une borne inférieure, notée α_u, pour le temps de cycle le CJSP associé. Cette borne peut être calculée comme suit :*

$$\alpha_u = \max_{j \in \mathcal{J}} \{ \frac{\sum_{J(i)=j} p_i}{H_{es}} \}.$$

où $\mathcal{J}$ représente l'ensemble des travaux et $J(i)$ correspond au travail contenant la tâche i.

Preuve :

Le CJSP, dans sa version relaxée (sans les contraintes de ressources) correspond à un GBCSP. Par définition du graphe associé au CJSP (voir 2.2.2), on a :

$$t_s(k) \geq -\alpha H_{es} + t_e(k)$$

ainsi,

$$\alpha \geq \frac{t_e(k) - t_s(k)}{H_{es}}. \tag{3.3}$$

Par ailleurs, le graphe du CJSP donne :

$$t_e(k) \geq (\sum_{J(i)=j} p_i) + t_s(k), \qquad \forall j \in \mathcal{J}$$

D'où, l'occurrence $< e, k >$ commence quand toutes les k occurrences de tous les travaux sont finies. On obtient ainsi :

$$t_e(k) \geq \max_{j \in \mathcal{J}} \{ \sum_{J(i)=j} p_i \} + t_s(k)$$

de plus,

$$t_e(k) - t_s(k) \geq \max_{j \in \mathcal{J}} \{ \sum_{J(i)=j} p_i \}$$

En tenant compte de l'équation (3.3), on a : $\alpha \geq \alpha_u$ avec

$$\alpha_u = \max_{j \in \mathcal{J}} \{ \frac{\sum_{J(i)=j} p_i}{H^*} \}.$$

□

Par ailleurs, le théorème 3.7, nous permet de calculer la borne inférieure du temps de cycle d'un CJSP.

Théorème 3.7. *On considère un CJSP et on note α^- la borne inférieure du temps de cycle.*

$$\alpha^- = \max\{\alpha_u, \alpha_s\}. \tag{3.4}$$

où α_u est la solution du GBCSP (CJSP sans les contraintes disjonctives) et α_s qui est définie comme suite :

$$\alpha_s = \max_{m \in \mathcal{M}} \{ \sum p_i | M(i) = m \}.$$

Preuve : La première borne α_u est donnée par le théorème 3.6. La deuxième borne α_s est la somme de toutes les durées de tâches utilisant la même machine, nous avons aussi :

$$t_j(k+1) \geq \sum_{M(i)=M(j)} p_i + t_j(k).$$

On en déduit que le temps de cycle ne peut pas être inférieur à la somme maximale des durées de tâches sur une même machine.

□

Borne supérieure du Work-In-Process

Dans le paragraphe 2.2.2, la notion de Work-In-Process (WIP) est introduite. Cette hauteur H_{es} de l'arc qui relie le noeud "e" au noeud "s" du graphe associé à un CJSP, correspond au nombre maximal de travaux dans un même cycle. Le fait d'augmenter le WIP permet d'influencer le temps de cycle optimal d'un CJSP, en effet, augmenter le WIP permet à des occurrences de tâches qui diffère d'au plus $H_{es} - 1$ d'être exécutées dans le même cycle, ce qui peut éventuellement réduire le temps de cycle optimal.

L'influence de la valeur du WIP sur le temps de cycle optimal du CJSP de l'exemple 2.5 peut être constatée : La figure 2.5 représente une solution du CJSP avec un WIP égal à 1, le temps de cycle de cette solution est de 11 unités de temps. Dans la figure 2.6, une solution de ce même CJSP est représentée mais avec un WIP égal à 2, le temps de cycle est passé à 7 unités de temps, puisqu'il est possible, par exemple, que l'occurrence $< 1, 1 >$ soit exécutée dans le même cycle que l'occurrence $< 2, 0 >$.

A quel moment l'augmentation du WIP n'a plus d'influence sur le temps de cycle optimal ? En se basant sur un résultat de Chauvet et al. [FCP03] qui donne une condition suffisante pour l'optimalité d'un ordonnancement cyclique, une borne supérieure du WIP est ainsi proposée.

Remarque 3.8. [FCP03] Soient $J \in \{J_1, J_2, ..., J_n\}$ un travail et $T(J)$ le temps de cycle d'une occurrence de J, autrement dit, le temps total nécessaire pour l'exécution d'une occurrence du travail J selon l'ordonnancement considéré. Le WIP H_{es} correspond à :

$$H_{es} = \frac{T(J_1) + T(J_2) + ... + T(J_n)}{C^*} \tag{3.5}$$

Où C^* est le temps de cycle de l'ordonnancement considéré.

Résultat 3.9. [FCP03] Soient $J \in \{J_1, J_2, ..., J_n\}$ un travail, C^* le temps de cycle de l'ordonnancement considéré, $T(J)$ le temps de cycle d'une occurrence de J et $P(J)$ la somme des durées de toutes les tâches de J. Si pour un ordonnancement donné,

$$\forall J \in \{J_1, J_2, ..., J_n\}, \quad \frac{T(J)}{C^*} \leq \left\lceil \frac{P(J)}{C^*} \right\rceil \tag{3.6}$$

alors l'ordonnancement est optimal, dans le sens où le maximum de productivité est réalisé avec le minimum de WIP. $\lceil a \rceil$ est le plus petit entier supérieur ou égal à a.

La somme sur les n travaux d'un CJSP de l'équation (3.6) donne l'inégalité suivante :

$$\sum_{k=1}^{n} \frac{T(J_k)}{C^*} \leq \sum_{k=1}^{n} \frac{P(J_k)}{C^*}$$

La première somme correspond bien au WIP, donc :

$$H_{es} \leq \sum_{k=1}^{n} \frac{P(J_k)}{C^*}$$

Si α^- est la borne inférieure du temps de cycle d'un CJSP définie par l'équation (3.4),

quelque soit l'ordonnancement considéré $\alpha^- \leq C^*$. Par conséquent, nous pouvons déduire une borne supérieure du WIP H_{es} d'un CJSP :

$$H_{es} \leq \sum_{k=1}^{n} \frac{P(J_k)}{\alpha^-} \tag{3.7}$$

3.1.2 Structure de la procédure de branch and bound

Pour avoir un moyen efficace de calcul de bornes pour tous les K_{ij}, une matrice des hauteurs B de taille $n \times n$ est construite. Ses coefficients B_{ij} sont initialisés par les H_{ij} pour les arcs uniformes et par les K_{ij} pour les arcs disjonctifs déjà déterminés. Par la suite, l'algorithme de Floyd et Warshall (voir [CLRS09]) de compléxité $O(n^3)$ est utilisé pour trouver les chemins les plus courts dans B entre toute paire de tâches. Ensuite les bornes inférieures K_{ij}^- sont déduite comme suit : $K_{ij}^- = 1 - B_{ji}$.

La mise à jour la matrice des hauteurs fait appel à l'algorithme de Floyd et Warshall, elle est donc susceptible de ralentir la procédure. Ainsi deux versions de l'algorithme sont proposées : Version 1 qui met à jour les bornes des K_{ij} en chaque noeud de l'arbre de recherche et par conséquent, n'examine pas la faisabilité de la solution. Version 2 qui ne calcule les bornes qu'une seule fois, lorsque l'algorithme est initialisé, mais vérifie la faisabilité de la solution lors de l'évaluation du noeud avec l'algorithme de Howard.

Dans un premier temps, la PSE version 1 est détaillée. Ensuite, la PSE version 2 est décrite au travers des différences entre les deux versions. L'algorithme peut être divisé en deux parties, à savoir la partie "Initialisation" où on initialise la procédure et la partie "Boucle" qui contient la boucle principale.

La partie "Initialisation" de l'algorithme est composée de la méthode *initialize()*, qui détermine la borne supérieure du temps de cycle en additionnant toutes les durées de tâches. Elle crée un premier noeud, calcule son temps de cycle et vérifie la faisabilité de la solution initiale. La borne inférieure du temps de cycle est calculée selon le théorème 3.7. Si le noeud initial est réalisable, il est empilé et l'algorithme détermine les bornes inférieures des décalages événementiels en deux temps :

- l'algorithme initialise tous les B_{ij} par H_{ij} si l'arc (i, j) existe et par ∞ sinon.
- l'algorithme de Floyd et Warshall est utilisé pour trouver le plus court chemin dans B.

La partie "Boucle" de l'algorithme répéte un ensemble d'opérations jusqu'à ce que la pile soit vide ou jusqu'à ce que borne inférieure soit égale à la borne supérieure pour le temps de cycle. La méthode *checkSum()* fixe les deux décalages événementiels $K_{ij} = K_{ij}^-$ et $K_{ji} = K_{ji}^-$ si $K_{ij}^- + K_{ji}^- = 1$. Si tous les K_{ij} sont déterminés, un ordonnancement complet est obtenu. Dans ce cas, la borne supérieure du temps de cycle est mise à jour par le α en cours si $\alpha \leq \alpha^+$. Si $\alpha \geq \alpha^+$, on coupe l'arbre de recherche. Dans le cas d'ordonnancement non complet, un nouveau K_{ij} est sélectionné selon la méthode *branchingRule()*. Cette méthode choisit un K_{ij} non déterminé et pour lequel $K_{ij}^- + K_{ji}^-$ est maximale, ce qui induit le nombre minimal de noeuds fils, et donc réduit l'arbre de recherche. L'algorithme fait ensuite appel à la méthode *branch()* qui crée un nouveau noeud fils pour tout entier dans l'intervalle $[K_{ij}^-, 1 - K_{ji}^-]$ pour le K_{ij} sélectionné. La méthode *evaluateNodes()* met

à jour les bornes inférieurs des décalages événementiels en insérant la valeur K_{ij} dans la matrice des hauteurs B et en faisant appel à l'algorithme de Floyd and Warshall, la deuxième étape de la méthode *evaluateNodes()* consiste à utiliser l'algorithme de Howard pour mettre à jour le temps de cycle. Enfin, l'algorithme appelle la méthode *nodeSelection()* qui empile tous les noeuds fils dans la pile selon la règle de sélection qui stipule que le noeud qui possède le plus petit temps de cycle est choisi en premier. En cas d'égalité de temps de cycle, l'algorithme classe les noeuds selon leurs indices. La procédure décrite peut également être vu dans la figure 3.2.

```
   Input: G = (T, E), affectation des tâches aux machines
   Output: α
 1 //Initialisation
 2 S_0 ← Initialize()
 3 //Boucle
 4 while PileNoeuds ≠ VIDE and α⁻ ≠ α⁺ do
 5 |   S ← Le premier élément de PileNeouds
 6 |   checkSum(S)
 7 |   if S(α) < α⁺ then
 8 |   |   if S est une séquence complète then
 9 |   |   |   α⁺ ← S(α)
10 |   |   else
11 |   |   |   S(K_ij sélectionné) ← branchingRule(S)
12 |   |   |   N ← branch(S)
13 |   |   |   evaluateNodes(N)
14 |   |   |   PileNoeud ←nodeSelection(N)
15 return α⁺
```

FIGURE 3.2 – Algorithme de la nouvelle procédure

Dans la version 2 de l'algorithme, C'est la méthode *evaluateNodes()* qui change : Tout d'abord, la matrice B n'est pas mis à jour, elle est seulement initialisée comme dans la version 1. Celà induit le deuxième ajustement : la faisabilité de chaque noeud est vérifiée avec l'algorithme de Howard.

3.1.3 Application de la procédure de branch and bound

Une résolution de l'exemple 2.5 avec la procédure version 1 est faite dans ce paragraphe.

La méthode *initialize()* calcule $\alpha^+ = 14$. Elle crée donc le premier noeud S_0 qui représente le graphe uniforme et calcule $\alpha^- = \max\{5; 7\} = 7$. Avant d'empiler S_0, les bornes inférieures de K_{13}, K_{31}, K_{24} et K_{42} sont calculées avec la matrice des hauteurs B est la suivante :

$$B = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 2 & 2 & 0 \\ 2 & 2 & 1 & 2 & 2 & 0 \\ 2 & 2 & 2 & 1 & 0 & 0 \\ 2 & 2 & 2 & 2 & 1 & 0 \\ 2 & 2 & 2 & 2 & 2 & 1 \end{pmatrix}.$$

TABLE 3.1 – La matrice des hauteurs B initiale

On sait que $K_{ij}^- = 1 - B_{ji} =$. Alors $K_{13}^- = -1$, $K_{31}^- = -1$, $K_{24}^- = -1$, et $K_{42}^- = 1$.

Dans le corps de l'algorithme, on sélectionne S_0 et on l'enlève de la pile de noeuds. La méthode *checkSum()* ne modifie pas la solution car $K_{13}^- + K_{31}^- \neq 1$ et $K_{24}^- + K_{42}^- \neq 1$. Pour S_0, $\alpha = 7$ donc $\alpha \leq \alpha^+$. La méthode *branchingRule()* sélectionne K_{24} plutôt que K_{13} car l'intervalle pour K_{13} est $[-1; 2]$ comparé à $[-1; 0]$. La méthode *branch()* crée deux noeuds S_1 et S_2 avec respectivement les valeurs ($K_{24} = -1$, $K_{42} = 2$) et ($K_{24} = 0$, $K_{42} = 1$). Pour les deux noeuds, la méthode *evaluateNodes()* calcule le temps de cycle, respectivement 7 et 9. Ensuite, elle met à jour K_{13}^- et K_{31}^-. Enfin, la méthode *nodeSelection()* choisit S_1. La deuxième itération continue avec S_1. Encore une fois, *checkSum()* ne modifie pas la solution. On choisit K_{13} pour brancher. Ensuite, la méthode *branch()* crée quatre noeuds fils S_3, S_4, S_5 et S_6 avec les valeurs $-1, 0, 1$ et 2 pour K_{24} et *evaluateNodes()* calcule leurs temps de cycle $11, 7, 7$ et 10. *nodeSelection()* choisit $S4$. A la troisième itération, on trouve une solution complète avec $\alpha \leq \alpha^+$ donc on remet à jour $\alpha^+ = \alpha = 7$. Ensuite, la procédure s'arrête car $\alpha^- = \alpha^+$ et le temps de cycle optimal 7 est trouvé.

La figure 3.3 affiche l'arbre de recherche pour l'exemple.

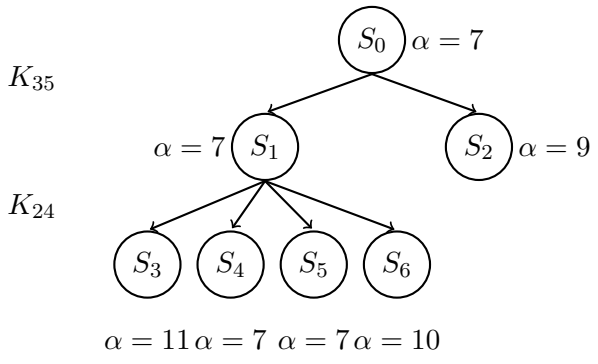

FIGURE 3.3 – Arbre de recherche de la PSE basée sur la nouvelle procédure pour l'exemple 2.5

3.2 Procédure de séparation et d'évaluation de Hanen pour le CJSP

Dans ce paragraphe, une décription de la procédure de branch and bound proposée par Hanen dans [Han94] est faite. Comme dans la partie 3.1, les concepts théoriques sous-jacents sont d'abord établis avant de décrire la structure de l'algorithme et de l'illustrer par l'utilisation de l'exemple 2.5.

3.2.1 Concepts théoriques

La Procédure de séparation et d'évaluation proposée par Hanen pour le CJSP est également basée sur la représentation par les graphes du CJSP, elle utilise deux concepts :

- Les fonctions de hauteur
- Les amplitudes

Définition 3.10. Soit $G = (T, E)$ le graphe associé à un CJSP, on note e_{ij} un arc uniforme de G et x_{ij} un arc disjonctif de G. Soit $HX : E \to \mathbb{Z}$ une fonction de hauteur pour le graphe G. HX est dite conservatrice si elle vérifie les deux propriétés suivantes :

$$\forall e_{ij} \in E, \quad HX(e_{ij}) = H_{ij} \tag{3.8}$$

$$\forall x_{ij} \in E, \quad HX(x_{ij}) + HX(x_{ji}) = 1. \tag{3.9}$$

A partir de la définition d'un MLP présenté dans la partie 2.2.4, Hanen associe une fonction de hauteur conservatrice à un ordonnacement cyclique faisable $S = (S_g, \alpha)$ tel qu'il est défini dans 2.3 : en définissant, le débit $\tau = \alpha^{-1}$, $u_i = \tau \times t_i$ pour toute tâche générique i ; pour chaque arc uniforme e_{ij} on pose $HX(e_{ij}) = H_{ij}$; et pour chaque arc disjonctif x_{ij} on pose $HX(x_{ij}) = \lceil u_i - u_j + \tau \times p_i \rceil$.

La notion d'amplitude, qui correspond aux parties à droite des inégalités du MLP, va permettre de caractériser un ordonnancement cyclique réalisable d'un CJSP.

Définition 3.11. Soient $G = (T, E)$ le graphe associé à un CJSP, τ la valeur du débit et HX une fonction de hauteur conservatice. L'amplitude d'un arc e quant à τ est définie comme suit :

$$\forall e \in E, \quad A_\tau(e) = \tau \times L(e) - HX(e). \tag{3.10}$$

Si μ est un chemin dans G, l'amplitude $A_\tau(\mu)$ est égale à la somme des amplitudes de tous les arcs du chemin μ.

Le théorème suivant donne une condition nécessaire à la faisabilité d'un ordonnancement cyclique, basée sur la notion de hauteur conservatrice.

Théorème 3.12. *Si HX est la fonction de hauteur associée à un ordonnancement cyclique réalisable $S = (S_g, \alpha)$, alors HX est conservatrice et tous les circuits de G ont des amplitudes quant à τ qui sont négatives ou nulles.*

Définition 3.13. Une fonction de hauteur HX est dite *réalisable quant à un $\tau > 0$ donné*, si HX est conservatrice et si tous les circuits de G ont des amplitudes quant à τ qui sont négatives ou nulles. HX est dite *faisable* si il existe un seul $\tau > 0$ tel que HX est faisable quant à $\tau > 0$.

Le théorème qui suit montre qu'une fonction de hauteur faisable quant à τ fournit un ordonnancement cyclique faisable de débit au plus égal à τ.

Théorème 3.14. *Soient $\tau^- \in \mathbb{R}^+$, HX une fonction de hauteur réalisable quant à τ^-. Pour tout $\alpha \geq 1/\tau^-$, il existe un ordonnancement cyclique réalisable $S = (S_g, \alpha)$ associé à HX. De plus, cet ordonnancement peut être calculer en $O(n^3)$.*

Résultat 3.15. Soit HX une fonction de hauteur conservatrice. Si un circuit de G a une hauteur négative ou nulle, alors HX n'est pas réalisable, sinon HX fournit un sous-ensemble non vide d'ordonnancements cycliques. L'ordonnancement optimal dans ce sous-ensemble peut être calculer en $O(n^3 log(n))$. Son temps de cycle correspond à la valeur du circuit critique du graphe G.

Le principe de l'algorithme de Hanen est de trouver des bornes sur les hauteurs pour que la fonction de hauteur reste réalisable, quant à un débit donné, dans des intervalles par dichotomie. Pour celà, il faut connaître l'amplitude d'un chemin de i à j et de j à i. En réalité, on peut se contenter d'une borne inférieure de l'amplitude maximale.

Définition 3.16. Soient $\tau \in \mathbb{R}^+$, HX une fonction de hauteur réalisable quant à τ. On définit l'amplitude maximale $\bar{A}_\tau$ quant à τ par la fonction suivante :

$$\bar{A}_\tau : T \times T \to \mathbb{Q} \cup \{-\infty\}$$

$$\bar{A}_\tau(i,j) = Max\left\{A_\tau(\mu) \ / \ \mu \ un \ chemin \ de \ i \ á \ j \ dans \ G\right\}. \tag{3.11}$$

S'il n'y a pas de chemin de i à j, $\bar{A}_\tau(i,j) = -\infty$.

Lemme 3.17. *Soit $\tau \in \mathbb{R}^+$ la valeur d'un débit. On définit pour chaque couple de tâches (i,j),*

$$\bar{A}^-(i,j) = Max\left\{A_\tau(\mu) = \tau \times L(\mu) - H(\mu) \ / \ \mu \ un \ chemin \ de \ i \ á \ j \ dans \ G\right\}. \tag{3.12}$$

Pour toute fonction de hauteur HX réalisable quant $\tau' \leq \tau$, $\bar{A}^-$ est une borne inférieure de l'amplitude maximale $\bar{A}_\tau$ associée à HX.

Ces résultats permettent, dans certains cas, de déterminer s'il existe une fonction de hauteur réalisable quant à une valeur de débit donnée τ. Grâce au Lemme 3.17, on peut calculer une borne inférieure $\bar{A}^-$ de l'amplitude maximale, qui permet de deduire la borne supérieure HX^+.

$$HX^+(x_{ij}) = \left\lfloor 1 - \tau \times L(x_{ij}) - \bar{A}^-_\tau(i,j) \right\rfloor. \tag{3.13}$$

Théorème 3.18. *Soit HX^+ une fonction de hauteur définie sur un graphe G. Soit V l'ensemble des fonctions de hauteur faisables, pour lesquelles HX^+ est une borne supérieure. Si un circuit de G a une hauteur HX^+ négative ou nulle, alors V est vide. Sinon, la temps de cycle α^- du circuit critique de G avec une fonction de longueur L et la fonction de hauteur HX^+ est une borne inférieure du temps de cycle pour tout ordonnancement faisable associé à une fonction de hauteur HX dans V.*

Ce Théorème 3.18 permet de mettre à jour la borne inférieure $\bar{A}^-$.

Algorithme de Gondran et Minoux

L'algorithme de Gondran et Minoux (voir [GM95]) utilisé dans la méthode proposée par Hanen, a une complexité de $O(n^3 log(n))$ pour déterminer le circuit critique d'un graphe, il est basé sur une dichotomie en $O(log(n))$ et un algorithme de recheche du plus court chemin comme l'algorithme de Bellman-Ford qui lui est en $O(n^3)$. Cet algorithme est aussi décrit dans [RV10].

Un pseudo code de l'algorithme de Gondran et Minoux est représenté dans la figure 3.4. L'algorithme initialise d'abord les bornes inférieure et supérieure du temps de cycle, dans la partie "Initialisation" de l'algorithme. Puisque le temps de cycle ne peut pas être négatif, La borne inférieure α^- est initialisée par 0. La borne supérieure α^+ est égale à la somme des durées de toutes les tâches calculée par la fonction *computeSumProcessingTimes()*. Une boucle est ainsi répétée jusqu'à ce que la borne inférieure α^- soit égale à la borne supérieure α^+ : L'algorithme calcule la valeur w qui permet de couper en deux l'intervalle $[\alpha^-, \alpha^+]$, puis évalue le graphe G en attribuant à chaque arc (i, j) le poids $L_{ij} - wH_{ij}$ et en utilisant l'algorithme de Bellman-Ford. Si le graphe ne contient aucun circuit de valeur positive (somme des poids de tous les arcs du circuit), la borne supérieure α^+ prend la valeur de w. Quand un circuit C positif de hauteur également positive est trouvé, la borne inférieure α^- reçoit la valeur du temps de cycle du circuit trouvé $\alpha(C)$. Si le circuit positif trouvé a une hauteur nulle, l'algorithme s'arrête car la solution n'est pas réalisable. Le dernier cas est celui où le circuit positif trouvé C a une hauteur négative, la borne supérieure α^+ reçoit la valeur du temps de cycle du circuit trouvé $\alpha(C)$. Dans la dernière partie, l'algorithme compare à nouveau les bornes du temps de cycle, si $\alpha^- = \alpha^+$, la solution optimale est trouvée.

3.2.2 Structure de la procédure de branch and bound de Hanen

Un pseudo code de l'algorithme de Hanen est présenté dans la figure 3.5. La procédure de Hanen nécessite pour commencer, une borne supérieure du temps de cycle, cette borne correspond à la somme des durées de toutes les tâches du CJSP dans un premier temps, en suite elle est améliorée si possible par une heuristique. L'initialisation de l'algorithme de Hanen appelle une méthode *Initialize()* qui crée noeud inital S_0. La boucle est répétée jusqu'à ce que la pile de noeuds soit vide ou que la borne inférieure du temps de cycle α^- soit égale à la borne supérieure α^+, dans cette partie, le premier noeud S de la pile est sélectionné puis évalué par la méthode *adjustmentProcedure()*. L'algorithme de Hanen

```
   Input: G = (T, E)
   Output: α
 1 //Initialisation
 2 α⁻ ← 0
 3 α⁺ ← computeSumProcessingTimes()
 4 //Boucle
 5 while α⁻ < α⁺ do
 6     w ← (α⁻+α⁺)/2
 7     (∗) ← evaluate(G, L − wH)
 8     if (∗) pas de circuit C de valeur positive then
 9         α⁺ ← w
10     else
11         if (∗) circuit positif C avec hauteur positive then
12             α⁻ ← α(C)
13         else
14             if (∗) circuit positif C avec hauteur nulle then
15                 return INFAISABLE
16             else
17                 (∗) retourne un circuit positif C avec hauteur négative
18                 α⁺ ← α(C)
19 //Conclusion
20 if α⁻ > α⁺ then
21     return INFAISABLE
22 else
23     return α⁻
24 return α
```

FIGURE 3.4 – Algorithme de Gondran et Minoux

effectue alternativement des calculs et des mises à jour des bornes HX^+ et $\bar{A}^-$, il s'arrête quand l'une des trois situations suivantes est vérifiée :

- Un circuit d'amplitude positive est détecté.
- La fonction de hauteur HX^+ devient conservative.
- Plus de mise à jour possible pour HX^+ ou $\bar{A}^-$.

Pendant l'exécution de la méthode *adjustmentProcedure()*, la borne inférieure $\bar{A}^-$ ne peut qu'augmenter, tandis que la borne supérieure HX^+ ne peut que diminuer. Les bornes inférieures des amplitudes $\bar{A}^-(i,j)$ sont stockées dans une matrice $n \times n$. Pour mettre à jour cette matrice, l'algorithme de Floyd et Warshall (voir [CLRS09]) de compléxité $O(n^3)$ est utilisé pour trouver les chemins les plus longs entre toute paire de tâches.

La méthode *checkSum()* fixe les hauteurs HX_{ij} et HX_{ji} aux valeurs de leurs bornes supérieures si leur somme est égale à 1, autrement dit : si $HX^+_{ij} + HX^+_{ji} = 1$, alors $HX_{ij} = HX^+_{ij}$ et $HX_{ji} = HX^+_{ji}$. Une séquence complète implique que tous les HX_{ij} sont déterminés, quand une séquence complète réalisable est obtenue, le circuit critique de cette solution est calculé $S(\alpha)$ avec la méthode *computeCriticalCircuit()* qui est une

```
Input: G = (T, E), affectation des tâches aux machines, α+
Output: α
1  //Initialisation
2  S0 ← Initialize()
3  //Boucle
4  while PileNoeuds ≠ VIDE and α− ≠ α+ do
5      S ← Le premier élément de PileNeouds
6      adjustmentProcedure(S)
7      checkSum(S)
8      if Pas de circuit C avec A(C) > 0 pour S then
9          if S est une séquence complète then
10             S(α) ← computeCriticalCircuit()
11             if S(α) < α+ then
12                 α+ ← S(α)
13         else
14             if S n'a plus d'intervalle pour tout Kij then
15                 S(selectedHXij) ← branchingRule(S)
16             N ← branch(S)
17             nodeStack ← nodeSeletion(N)
18 return α
```

FIGURE 3.5 – Algorithme de Hanen

implémentation de l'algorithme de Gondran et Minoux. Par la suite la borne supérieure du temps de cycle α^+ reçoit la valeur $S(\alpha)$ si $S(\alpha) < \alpha^+$. Si la séquence n'est pas complète, l'algorithme appelle la méthode *branchingRule()* pour déterminer un nouveau décalage événementiel qui va être fixé. Quand un décalage événementiels HX_{ij} est sélectionné, deux arcs disjonctifs sont ajoutés au graphe. Chacun de ces deux arcs crée au moins un nouveau circuit dans le graphe. Pour qu'un décalage événementiels HX_{ij} soit sélectionné par la méthode *branchingRule()*, il faut que les bornes inférieures des amplitudes des deux circuits soient minimales. Cependant, cette méthode n'est utilisée que quand le noeud S n'a plus d'intervalle pour tout K_{ij} mais des valeurs dans (Z). Dans la méthode *branch()*, une dichotomie est faite : Chaque intervalle $\left[1 - HX^+_{ji}, HX^+_{ij}\right]$, correspondant à une hauteur HX_{ij}, est découpé en deux parties égales, ce qui permet à l'algorithme de créer deux noeuds fils à chaque branchement.

3.2.3 Application de la procédure de branch and bound de Hanen

Dans ce paragraphe, une illustration de la procédure de Hanen est fait à travers l'exemple 2.5. En plus des données du CJSP représentées dans le tableau 2.1 qui permettent d'établir le graphe uniforme associé au CJSP et d'y ajouter les différents arcs disjonctifs, la méthode de Hanen requiert une borne supérieure du temps de cycle $\alpha^+ = 10$ selon l'heuristique prévue pour cet effet. L'arbre de recherche est initialisé par un premier noeud S_0, ce noeud est ensuite donné comme paramètre à la méthode *adjustmentProcedure()*, qui

initialise selon l'équation (3.12), la matrice des amplitudes est la suivante.

$$\bar{A}^- = \begin{pmatrix} 0 & 0 & 0.5 & 0 & 0.2 & 0.9 \\ -1.1 & 0 & 0.5 & -1.1 & -0.9 & 0.9 \\ -1.6 & -1.6 & 0 & -1.6 & -1.4 & 0.4 \\ -1.5 & -1.5 & -1 & 0 & 0.2 & 0.5 \\ -1.7 & -1.7 & -1.2 & -1.7 & 0 & 0.3 \\ -2 & -2 & -1.5 & -2 & -1.8 & 0 \end{pmatrix}.$$

Par exemple, la borne $\bar{A}^-(0, t_2)$ est calculée de la manière suivante :

$$\bar{A}^-(0, t_2) = \frac{1}{\alpha^+} \times (L_{01} + L_{12}) - (H_{01} + H_{12}) = \frac{1}{10} \times (0 + 5) - (0 + 0) = 0.5.$$

La méthode *adjustmentProcedure()* initialise également les bornes HX^+ en utilisant l'équation 3.13, ce qui donne : $HX^+_{13} = \left\lfloor 1 - \frac{1}{10} \times 2 - (-1, 1) \right\rfloor = 1$, $HX^+_{31} = 2$, $HX^+_{24} = 2$ et $HX^+_{42} = 1$. Puisque $HX^+_{13} + HX^+_{31} \neq 1$ et $HX^+_{24} + HX^+_{24} \neq 1$, la méthode *checkSum()* ne modifie rien. Comme il n'y a pas de circuit C avec une amplitude $A(C) > 0$ pour S_0, l'algorithme choisit une paire d'arcs disjonctifs pour créer deux noeuds fils. La méthode *branchingRule()* a le choix entre (x_{13}, x_{31}) ou (x_{24}, x_{42}) , les amplitudes maximales des circuits passant par les quatre arcs disjonctifs sont calculées dans l'ordre : -1, -0.9, -1.8 et -0.1. L'arc x_{24} correspond au minimum des amplitudes maximales, il est donc sélectionné. Par conséquent, la méthode *branch()* crée deux noeuds fils S_1 et S_2, le noeud S_1 correspond à l'intervalle $[0, 1]$ pour la hauteur HX_{24} et le noeud S_2 correspond à la valeur 2 pour la hauteur HX_{24} (car S_0 correspondait à l'intervalle $[0, 2]$ pour la hauteur HX_{24}). Puisque le noeud fils S_0 affiche la plus petite valeur des amplitudes maximales de circuits, alors l'algorithme va poursuivre avec ce noeud.

Au cours de la deuxième itération de la boucle de l'algorithme, les méthodes *adjustmentProcedure()* et *checkSum()* ne modifient rien aux valeurs des différentes variables, comme il n'y a ni circuit C avec une amplitude $A(C) > 0$ ni séquence complète pour S_1. La méthode *branch()* crée à nouveau deux noeuds fils : S_3 et S_4, le noeud S_3 correspond à la valeur 0 pour la hauteur HX_{24} et le noeud S_4 correspond à la valeur 1 pour cette même hauteur. Leur amplitude maximale de circuit est calculée : -0.8 pour S_3 et -1.1 pour S_4. L'algorithme continue donc avec S_4.

Dans la troisième itération de la boucle de l'algorithme de Hanen, la méthode *adjustmentProcedure()* reçoit S_4 comme paramètre, elle met à jour les bornes HX^+ pour les décélages événementiels non fixés $HX^+_{13} = 1$ et $HX^+_{31} = 1$. La méthode *branchingRule()* choisit la paire d'arc (x_{13}, x_{31}) et crée deux noeuds : le noeud S_5 pour la valeur 0 de la hauteur HX_{13} et le noeud S_6 pour la valeur 1 pour cette même hauteur. Comme pour la précédente itération, les amplitudes maximales de circuits sont calculées : -0.6 pour S_5 et -0.9 pour S_6. L'itération suivante de l'algorithme reçoit une séquence complète, La méthode *computeCriticalCircuit()* calcule $S_6(\alpha) = 7$. Puisque $S_6(\alpha) < \alpha^+$, alors $\alpha^+ = 7$. Dans les itérations suivantes de la boucle de l'algorithme, S_5 est écarté car son temps de cycle est égal à 7, S_3 est évalué et deux autres noeuds sont créés puis écartés pour la même raison que S_5, S_2 n'est pas exploré puisqu'un circuit C d'amplitude $A(C) > 0$ est trouvé.

La procédure se finit avec un temps de cycle optimal $\alpha^+ = 7$. L'arbre de recherche de la procédure de Hanen pour l'exemple 2.5 est affiché dans la figure 3.6.

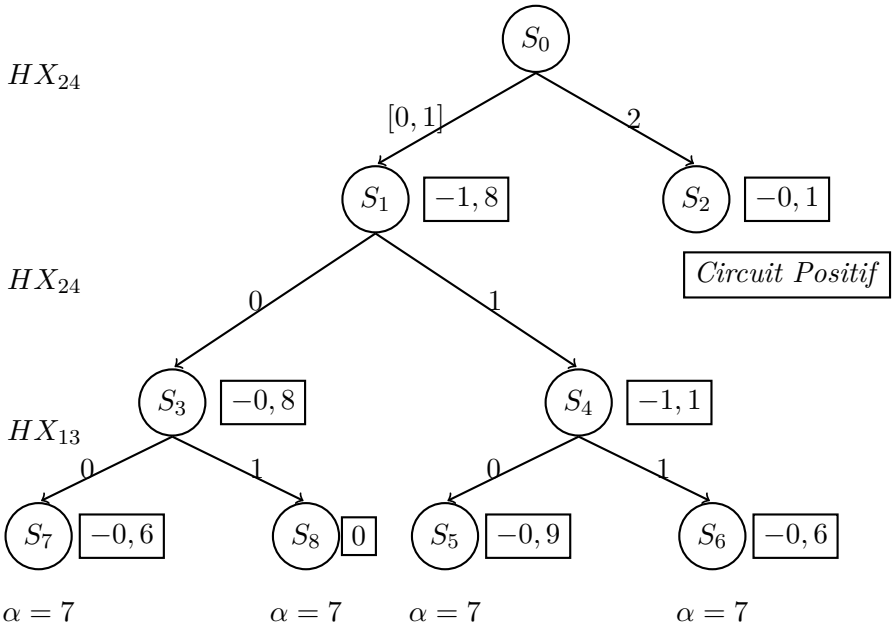

FIGURE 3.6 – Arbre de recherche de la PSE basée sur la procédure de Hanen pour l'exemple 2.5

Chapitre 4

Approche par la théorie des tas

Plusieurs approches ont été utilisées pour résoudre le problème d'odonnancement cyclique, elles ont fait l'objet du chapitre 2.2. Parmi les plus originales, on peut citer la méthode de [GM99]. Ce chapitre est concsacré à la présentation de cette méthode ainsi qu'à ses extensions, ce travail est aussi présenté dans [BHA10].

Les auteurs de [GM99] montrent que le sous-problème de l'évaluation d'une solution (donc un ordonnancement) peut être résolu aisément avec l'aide de la théorie des tas. Plus précisémment, l'évaluation du taux de production du cycle revient au calcul d'une valeur propre d'un produit de matrices dans lequel chacune des matrices représente une opération élémentaire. Cette propriété s'avère particulièrement intéressante dans le cas de l'évaluation successive d'un grand nombre d'ordonnancement. En outre, la théorie des tas permet une représentation graphique très intuitive d'un ordonnancement, puisque celui-ci s'illustre comme un empilement de plusieurs briques (en fait, un « tas » de briques) dont le contour supérieur fournit les dates au plus tôt des dernières opérations des machines. Plusieurs cycles seront alors symbolisés par l'empilement de plusieurs pièces formant un tas. Le tas formé n'est pas sans rappeler le célèbre jeu vidéo "Tetris" (même si les pièces sont parfois non connexes et surtout, la finalité totalement différente).

Dans ce chapitre, la première section fait l'objet de la présentation de cette théorie. Le passage d'un problème d'ordonnancement cyclique à un modèle de type tas est exposé dans une seconde partie. Une brève étude de la complexité est aussi présentée pour asseoir la pertinence de la méthode. La dernière partie est consacrée aux extensions de cette méthode afin de traiter des cas plus généraux d'ordonnancement cycliques.

4.1 Théorie des tas

La théorie des tas fournit un modèle algébrique et graphique à des problèmes d'ordonnancement ou plus généralement de séquencement. La partie algébrique est basée sur l'algèbre $(max, +)$, le modèle graphique décrit un tas où des pièces s'empilent les unes sur les autres. Une présentation plus complète de la théorie des tas est proposée dans [GM99].

Graphiquement, les modèles de type "tas de pièces" présentent un axe vertical indiquant la hauteur du tas. L'axe horizontal est quant à lui constitué d'un nombre fini de *slots*. Une pièce est un bloc solide, éventuellement non connecté, qui occupe une partie des slots

avec des contours inférieur et supérieur. A une séquence ordonnée de pièces est associé un tas correspondant à l'empilement des pièces. Une pièce occupe la position la plus basse possible par rapport à l'axe horizontal du tas et les pièces déjà empilées.

Exemple 4.1. L'exemple de la figure 4.1, où on doit empiler quatre pièces a, b, c et d, montre bien qu'en modifiant l'ordre d'empilement des pièces, on peut réduire la hauteur du tas. En effet, l'ordre $a \to b \to c \to d$ donne une hauteur du tas de pièces égale à 11, alors que l'ordre $a \to c \to b \to d$ donne une hauteur de 10.

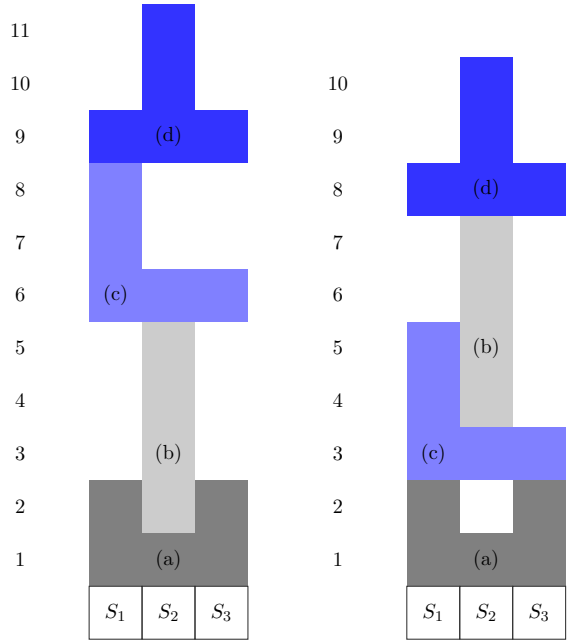

FIGURE 4.1 – Exemple d'un tas de pièces

Plus formellement, nous définissons un modèle de type tas ci-dessous.

Définition 4.2. Un modèle de type tas est un 5-uplet $H = (\mathcal{T}, \mathcal{R}, R, l, U)$, où

- $\mathcal{T}$ est un ensemble fini de pièces,
- $\mathcal{R}$ est un ensemble fini de slots,
- $R : \mathcal{T} \to \mathcal{P}(\mathcal{R})$ correspond au sous-ensemble de slots occupés par une pièce (avec $\mathcal{P}(\mathcal{R})$ l'ensemble des parties de $\mathcal{R}$).
- $l : \mathcal{T} \times \mathcal{R} \to \mathbb{R}_{max}$ donne la hauteur du contour inférieur d'une pièce dans les différents slots.
- $u : \mathcal{T} \times \mathcal{R} \to \mathbb{R}_{max}$ donne la hauteur du contour supérieur d'une pièce dans les différents slots.

Par convention, on a :
– $l(a, r) = u(a, r) = -\infty$ si $r \notin R(a)$;
– $min_{r \in R(a)} l(a, r) = 0$ (ce qui revient à dire qu'une pièce ne peut pas aller plus bas que le sol).
Chaque pièce a est représentée par sa matrice définie de la manière suivante :

$$\mathcal{M}(a)_{sr} = \begin{cases} 0, & si\ s = r,\ r \notin R(a) \\ u(a,r) - l(a,s), & si\ r \in R(a),\ s \in R(a) \\ -\infty, & sinon. \end{cases} \tag{4.1}$$

Le mot $w = a_1 a_2 \dots a_k \in \mathcal{T}^*$ de longueur k est interprété comme le tas obtenu en empilant les k pièces $a_1, a_2, \dots, a_k$ dans cet ordre. La matrice correspondante à ce tas est définie de la manière suivante :

$$\mathcal{M}(w) = \mathcal{M}(a_1 \dots a_k) = \mathcal{M}(a_1) \otimes \dots \otimes \mathcal{M}(a_k) \tag{4.2}$$

Le contour supérieur du tas w est le vecteur ligne $x_{\mathcal{H}}(w)$, de dimension $card(\mathcal{R})$ où $x_{\mathcal{H}}(w)_r$ est la hauteur du tas relativement au slot r tel que :

$$\forall w \in \mathcal{T}^* \quad x_{\mathcal{H}}(w) = x_{\mathcal{H}}(e)^t\ \mathcal{M}(w). \tag{4.3}$$

avec $x_{\mathcal{H}}(e)$ vecteur de dimension $card(\mathcal{R})$ composé de 0 (l'élément neutre de $\otimes$).

$$x_{\mathcal{H}}(e) = \mathbb{1}_{\mathbb{R}_{max}} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

La hauteur du tas de pièces est donnée par la relation suivante :

$$\forall w \in \mathcal{T}^* \quad y_{\mathcal{H}}(w) = \max_{r \in \mathcal{R}} x_{\mathcal{H}}(w)_r = x_{\mathcal{H}}(w)\ x_{\mathcal{H}}(e). \tag{4.4}$$

Exemple 4.3. Dans l'exemple 4.1, les matrices des pièces sont les suivantes :

$$\mathcal{M}(a) = \begin{pmatrix} 2 & 1 & 2 \\ 2 & 1 & 2 \\ 2 & 1 & 2 \end{pmatrix}, \quad \mathcal{M}(b) = \begin{pmatrix} 0 & \varepsilon & \varepsilon \\ \varepsilon & 4 & \varepsilon \\ \varepsilon & \varepsilon & 0 \end{pmatrix}$$

$$\mathcal{M}(c) = \begin{pmatrix} 3 & 1 & 1 \\ 3 & 1 & 1 \\ 3 & 1 & 1 \end{pmatrix}, \quad \mathcal{M}(d) = \begin{pmatrix} 1 & 3 & 1 \\ 1 & 3 & 1 \\ 1 & 3 & 1 \end{pmatrix}$$

L'équation (4.2) permet de calculer les matrices des deux tas de l'exemple 4.1.

$$\mathcal{M}(abcd) = \begin{pmatrix} 9 & 11 & 9 \\ 9 & 11 & 9 \\ 9 & 11 & 9 \end{pmatrix}, \quad \mathcal{M}(acbd) = \begin{pmatrix} 8 & 10 & 8 \\ 8 & 10 & 8 \\ 8 & 10 & 8 \end{pmatrix}$$

En effet, on obtient les contours supérieurs et les hauteurs de tas de pièces représentés en figure 4.1 :

$$x_{\mathcal{H}}(abcd) = (9\ 11\ 9) \quad \text{et} \quad x_{\mathcal{H}}(acbd) = (8\ 10\ 8)$$

$$y_{\mathcal{H}}(abcd) = 11 \quad \text{et} \quad y_{\mathcal{H}}(acbd) = 10.$$

4.2 Ordonnancement *1*-cyclique

La théorie des tas présentée dans le paragraphe 4.1 trouve un écho dans le domaine de l'ordonnancement cyclique. En effet, elle permet d'évaluer la performance d'un ordonnancement cyclique. Dans [GM99], les auteurs démontrent les liens entre la théorie des tas et les problèmes d'ordonnancement cycliques. En plus d'être graphiquement évidente, cette théorie permet de calculer la performance d'une certaine classe d'ordonnancement. Dans le cas d'ordonnacements 1-cyclique, ce calcul est équivalent au calcul classique d'un temps de cycle d'un ordonnancement.

4.2.1 Application de la théorie des tas à un problème d'ordonnancement *1*-cyclique

Construction du modèle de type Tas

La procédure initiée dans [GM99] permet, à partir des données d'un problème d'ordonnancement cyclique, de construire un modèle de type "Tas" dans lequel :

- $\mathcal{T}$ contient l'ensemble des tâches élémentaires. Chaque tâche élémentaire correspond à une pièce dont la forme est fonction des ressources utilisées et du travail auquel elle appartient.
- $\mathcal{R}$ l'ensemble de slots. Le nombre de slots du tas obtenu sera égal à la somme des ressources et des travaux à effectuer. Dans les CJSP 1-cyclique, une nouvelle occurrence d'un travail ne peut débuter qu'après la fin de l'occurrence précédente de ce même travail. Cette contrainte amène à prendre en compte les travaux en tant que slots dans notre modèle.
- $R : \mathcal{T} \to \mathcal{P}(\mathcal{R})$ le sous-ensemble de slots occupés par une pièce est composé de deux slots : le slot de la machine utilisée par la tâche élémentaire correspondante et le slot du travail auquel elle appartient.
- $l : \mathcal{T} \times \mathcal{R} \to \mathbb{R}_{max}$ la hauteur du contour inférieur d'une pièce dans les deux slots qu'elle occupe est égale à 0, sinon à ε.
- $u : \mathcal{T} \times \mathcal{R} \to \mathbb{R}_{max}$ la hauteur du contour supérieur d'une pièce dans les deux slots qu'elle occupe est égale à la durée de la tâche élémentaire correspondante, sinon à ε.

Cette modélisation du CJSP par un modèle de type Tas ne prend en considération que le cas des ordonnancements saufs c'est-à-dire dont le RdP associé est sauf (voir 1.21). Autrement dit, à tout moment dans un ordonnancement sauf, une et une seule occurrence d'un travail donné est en cours d'exécution. On peut aussi exprimer cette particularité par le fait que les travaux ne s'entrelacent pas. Ainsi le modèle présenté est associé à des ordonnancements saufs avec des solutions 1-cyclique.

Les dates de fin d'utilisation d'une machine ou les dates de fin d'un travail pour un cycle sont données par :

$$C = x_{\mathcal{H}}(e)^t \otimes \mathcal{M}. \tag{4.5}$$

avec $x_{\mathcal{H}}(e)$ vecteur de dimension $card(\mathcal{R})$ (la somme du nombre de machines et du nombre de travaux) défini dans 4.1, et $\mathcal{M}$ la matrice associée à l'empilement des pièces.

Le C_{max}, c'est-à-dire la plus grande date de fin des travaux, peut également être déterminé :

$$C_{max} = x_{\mathcal{H}}(e)^t \otimes \mathcal{M} \otimes x_{\mathcal{H}}(e). \tag{4.6}$$

Ce modèle rend aussi possible l'évaluation du temps de cycle d'un ordonnancement cyclique à travers le théorème suivant :

Théorème 4.4. *Le temps de cycle d'un ordonnancement cyclique $w = t_1 \dots t_n$ est donné par :*

$$\alpha_w = \rho(\mathcal{M}(t_1) \otimes \dots \otimes \mathcal{M}(t_n)). \tag{4.7}$$

où $\rho(\mathcal{X})$ désigne la valeur propre de la matrice $\mathcal{X}$ (voir §1.1)

Preuve : Considérons la séquence $t_1 \dots t_n$ dans un modèle de type Tas, on note v le tas formé par l'accumulation des pièces $t_1 \dots t_n$. v peut également être considérée comme une unique pièce dont la matrice $\mathcal{M}(v) = \mathcal{M}(t_1) \otimes \dots \otimes \mathcal{M}(t_n)$. L'évaluation du tas est alors donnée par l'empilement de la pièce v. Comme on l'a vu précédemment, le contour supérieur du tas est donné par : $x(v) = x_{\mathcal{H}}(e)^t \otimes \mathcal{M}(v)$ est donc, $x(v \dots v) = x_{\mathcal{H}}(e)^t \otimes \mathcal{M}(v) \otimes \dots \otimes \mathcal{M}(v)$. En prenant l'hypothèse que $\mathcal{M}(v)$ soit irréductible, ce qui est toujours le cas pour les problèmes de Job-Shop, un comportement cyclique apparaît (voir le théorème 1.15) après l'empilement d'un certain nombre de pièces.

On a donc, $x(v \dots v) = x_{\mathcal{H}}(e)^t \otimes \mathcal{M}(v)^{k+C} = x_{\mathcal{H}}(e)^t \otimes \rho(\mathcal{M}(v))^C \mathcal{M}(v)^k$.

Ainsi, $x(|v|_{k+C}) = \rho(\mathcal{M}(v))^C \otimes x(|v|_k)$.

□

Exemple 4.5. Si on considère les tâches génériques de l'exemple 2.5, les matrices correspondant à ces tâches $\mathcal{M}(t_i)$ seront :

$$\mathcal{M}(t_1) = \begin{pmatrix} 5 & \varepsilon & 5 & \varepsilon \\ \varepsilon & 0 & \varepsilon & \varepsilon \\ 5 & \varepsilon & 5 & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & 0 \end{pmatrix}, \quad \mathcal{M}(t_2) = \begin{pmatrix} 0 & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & 4 & 4 & \varepsilon \\ \varepsilon & 4 & 4 & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & 0 \end{pmatrix}$$

$$\mathcal{M}(t_3) = \begin{pmatrix} 2 & \varepsilon & \varepsilon & 2 \\ \varepsilon & 0 & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & 0 & \varepsilon \\ 2 & \varepsilon & \varepsilon & 2 \end{pmatrix}, \quad \mathcal{M}(t_4) = \begin{pmatrix} 0 & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & 3 & \varepsilon & 3 \\ \varepsilon & \varepsilon & 0 & \varepsilon \\ \varepsilon & 3 & \varepsilon & 3 \end{pmatrix}$$

On calcule la matrice produit de l'ordonnancement $w_0 = t_1 t_2 t_3 t_4$:

$$\mathcal{M}(w_0) = \begin{pmatrix} 7 & 12 & 9 & 12 \\ \varepsilon & 7 & 4 & 7 \\ 7 & 12 & 9 & 12 \\ 2 & 5 & \varepsilon & 5 \end{pmatrix}$$

Avec $\rho(\mathcal{M}(w_0)) = 9$ la valeur propre de la matrice $\mathcal{M}(w_0)$.

La figure 4.2 est une représentation du tas de pièces obtenu par l'ordonnancement w_0 des opérations de l'exemple 2.5. Le nombre de slots est de quatre : deux machines M_1 et M_2 et deux travaux T_1 et T_2.

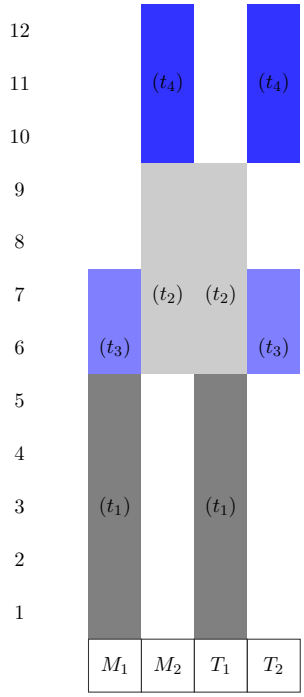

FIGURE 4.2 – Tas de pièces - Ordonnancement w_0

La valeur propre de la matrice $\mathcal{M}(w_0)$ correspond bien au temps de cycle de l'ordonnancement $w_0 = t_1t_2t_3t_4$.

Le produit de matrices tel qu'il est défini dans §4.1 à une compléxité en $O(n^4)$. Cependant, la structure très particulière des matrices de tâches (très creuses) permet un calcul du produit moins couteux. Plus précisémént, le produit d'une matrice quelconque par une matrice associée à une tâche peut être défini par la proposition suivante :

Propriété 4.6. *Soit t une tâche de durée p_t appartenant au travail j et utilisant la machine s. La matrice $\mathcal{M}(t)$ associée à t s'écrit :*

$$\mathcal{M}(t) = \begin{pmatrix} 0 & \cdots & \varepsilon & \cdots & \varepsilon & \varepsilon \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \varepsilon & \cdots & p_t & \cdots & p_t & \varepsilon \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \varepsilon & \cdots & p_t & \cdots & p_t & \varepsilon \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \varepsilon & \cdots & \varepsilon & \cdots & \varepsilon & 0 \end{pmatrix}.$$

Soit $\mathcal{M}$ une matrice quelconque, le produit $\mathcal{M}\mathcal{M}(t)$ se calcule comme suit :

– *Les colonnes* $C_i^{\mathcal{M}\mathcal{M}(t)}$ *de* $\mathcal{M}\mathcal{M}(t)$ *telles que* $i \neq j$ *et* $i \neq s$ *sont égales aux colonnes* $C_i^{\mathcal{M}}$ *de* $\mathcal{M}$.
– *Les colonnes* C_j *et* C_s *de* $\mathcal{M}\mathcal{M}(t)$ *sont égales à :*

$$C_j^{\mathcal{M}\mathcal{M}(t)} = C_s^{\mathcal{M}\mathcal{M}(t)} = \left(C_s^{\mathcal{M}} \oplus C_j^{\mathcal{M}}\right) \otimes p_t$$

Le calcul du produit de matrices selon la propriété 4.6 permet de réduire sa compléxité qui passe à $O(n)$.

Equivalence des ordonnancements

Dans cette partie, une propriété de commutation élémentaire est définie. Cette propriété permet d'éviter l'évaluation d'ordonnancements redondants et par la suite, d'élaguer efficacement l'arbre de recherche de la PSE basée sur la théorie des tas.

Propriété 4.7.

$$\forall a, b \in \mathcal{T} \qquad R(a) \cap R(b) = \emptyset \;\Rightarrow\; \mathcal{M}(a)\mathcal{M}(b) = \mathcal{M}(b)\mathcal{M}(a).$$

Preuve : On note j_a (resp. j_b) le travail auquel la tâche a (resp. b) appartient et s_a (resp. s_b) la machine sur laquelle est executée la tâche a (resp. b). Le calcul du produit $\mathcal{M}(a)\mathcal{M}(b)$ selon la propriété 4.6 donne :
– Les colonnes $C_i^{\mathcal{M}(a)\mathcal{M}(b)}$ telles que $i \neq j_b$ et $i \neq s_b$ sont égales aux colonnes $C_i^{\mathcal{M}(a)}$.
– Les colonnes C_{j_b} et C_{s_b} de $\mathcal{M}(a)\mathcal{M}(b)$ sont égales à :

$$C_{j_b}^{\mathcal{M}(a)\mathcal{M}(b)} = C_{s_b}^{\mathcal{M}(a)\mathcal{M}(b)} = \left(C_{s_b}^{\mathcal{M}(a)} \oplus C_{j_b}^{\mathcal{M}(a)}\right) \otimes p_b = C_{j_b}^{\mathcal{M}(b)} = C_{s_b}^{\mathcal{M}(b)}.$$

Donc, le produit $\mathcal{M}(a)\mathcal{M}(b)$ est égal à la matrice $\mathcal{M}(a)$ où les colonnes $C_{j_b}^{\mathcal{M}(a)}$ et $C_{s_b}^{\mathcal{M}(a)}$ sont remplacées respectivement par $C_{j_b}^{\mathcal{M}(b)}$ et $C_{s_b}^{\mathcal{M}(b)}$.

De la même manière, le produit $\mathcal{M}(b)\mathcal{M}(a)$ est égal à la matrice $\mathcal{M}(b)$ où les colonnes $C_{j_a}^{\mathcal{M}(b)}$ et $C_{s_a}^{\mathcal{M}(b)}$ sont remplacées respectivement par $C_{j_a}^{\mathcal{M}(a)}$ et $C_{s_a}^{\mathcal{M}(a)}$.

Sachant que $R(a) \cap R(b)$ c.à.d $j_a \neq j_b$ et $s_a \neq s_b$, alors : $\mathcal{M}(a)\mathcal{M}(b) = \mathcal{M}(b)\mathcal{M}(a)$.

□

Autrement dit, si deux tâches a et b n'utilisent pas la même machine et ne font pas partie du même travail, alors le produit de leur matrice associée est commutatif. On peut alors définir l'équivalence suivante entre deux ordonnancements.

Propriété 4.8. *Soient deux ordonnancements* $w_1 = uabv$ *et* $w_2 = ubav$ *dans lesquels les tâches* a *et* b *n'ont pas de slot en commun :* $R(a) \cap R(b) = \emptyset$. *Les ordonnancements* w_1 *et* w_2 *sont équivalents et on a l'égalité suivante* $\rho(\mathcal{M}(w_1)) = \rho(\mathcal{M}(w_2))$.

Preuve : Découle directement de la propriété 4.7.

□

4.2.2 Structure de la procédure de branch and bound pour la théorie des tas

PSE pour la théorie des tas

L'algorithme de séparation et d'évaluation basé sur la théorie des tas est décrit dans ce paragraphe. Cet algorithme peut également être divisé en deux parties, à savoir la partie "Initialisation" et la partie "Boucle".

L'initialisation de l'algorithme est composée de la méthode *initialize()*, qui détermine toutes les matrices associées aux différentes tâches en fonction du travail dont elles font partie et de la machine qu'elles utilisent. La borne supérieure α^+ du temps de cycle correspond à la valeur propre de la matrice M_{init} qui représente le produit de toutes les matrices associées aux tâches, ce produit est effectué selon l'ordre des tâches dans les travaux, ce qui revient à calculer le temps de cycle d'un ordonnancement réalisable. La borne inférieure α^- du temps de cycle est donnée par le 3.7. Concernant la borne α_u du 3.7, son calcul devient le maximum des sommes des temps d'éxecution par travail puisque $H_{es} = 1$ (voir 3.6).

La pile est initialisée avec la première tâche du premier travail. Cette initialisation arbitraire est sans conséquence puisque l'ordonnacement est cyclique. Plus précisément un ordonnancement avec un motif $t_1t_2t_3t_4$ qui se répète indéfinniment est équivalent à un ordonnancement de motif $t_2t_3t_4t_1$. Dans [GM99], ces ordonnancements sont qualifiés de conjugués cycliques.

Dans la partie "Boucle" de l'algorithme répéte un ensemble d'opérations jusqu'à ce que la pile soit vide ou jusqu'à ce que borne inférieure soit égale à la borne supérieure pour le temps de cycle. Pour un noeud N, pour toute une tâche, l'algorithme vérifie si cette tâche ne fait pas déjà partie du noeud courant N. Si cette tâche n'est pas encore affectée, il vérifie si elle représente le début d'un travail ou si elle succède à un tâche affectée dans le noeud N. Quand une tâche t est sélectionnée, un noeud fils N_{fils} est créé si un ordonnancement équivalent n'a pas déjà été précédemment empilé (voir §4.2.1).

Le noeud N_{fils} contient la même séquence de tâches que le noeud N augmentée de la tâche t et sa matrice $\mathcal{M}(N_{fils})$ correspond au produit de la matrice du noeud N par la matrice associée à la tâche t. Le temps de cycle α en cours est donné par la valeur propre de la matrice $\mathcal{M}(N_{fils})$. Si le temps de cycle α est inférieur à α^+, le noeud N_{fils} est empilé. L'algorithme vérifie par la suite si ce noeud N_{fils} contient toutes les tâches, si tel est le cas, un ordonnancement complet est obtenu, la borne supérieur α^+ est remplacée par α le temps de cycle en cours.

La procédure décrite est représenté dans la figure 4.3.

Application de la procédure de branch and bound

Dans ce paragraphe, une résolution de l'exemple 2.5 avec la PSE basée sur la théorie des tas est présentée.

La méthode *initialize()*, détermine toutes les matrices associées aux différentes tâches. La borne supérieure α^+ du temps de cycle est égale à 9, la valeur propre de la matrice M_{init}

Input: $G = (T, E)$, affectation des tâches aux machines
Output: α

```
1  //Initialisation
2  M_init ← Initialize()
3  α⁺ ← ρ(M_init)
4  α⁻ ← maxSumProcessingTimeMachine()
5  //Boucle
6  while PileNoeuds ≠ ∅ ET α⁻ ≠ α⁺ do
7      N ← Le premier élément de PileNoeuds
8      for t ∈ T do
9          if (t ∉ N.sequence) ET (t est un début de travail OU t est un successeur
           d'une tâche dans N.sequence) then
10             N_fils.sequence ← N.sequence + t
11             N_fils.matrice ← N.matrice⊗M(t)
12             α ← ρ(N_fils.matrice)
13             if α < α⁺ then
14                 PileNoeuds.push(N_fils)
15                 if longueur(N_fils.séquence) = nombre de tâches then
16                     α⁺ ← α
17 return α⁺
```

FIGURE 4.3 – Algorithme de séparation et d'évaluation pour la théorie des tas.

qui correspond à la matrice $\mathcal{M}(w_0)$ de l'exemple (4.5), l'ordonnancement initial réalisable est dans ce cas $w_0 = t_1t_2t_3t_4$. La somme des durées de tâche sur la machine 1 est égale à la somme des durées de tâche sur la machine 2. La borne inférieure α^- est donc égale à 7.

L'algorithme crée ensuite un premier noeud N avec la tâche t_1 puis les séquences t_1t_2 et t_1t_3.

La matrice du noeud t_1t_2 est

$$\mathcal{M}(t_1t_2) = \begin{pmatrix} 5 & 9 & 9 & \varepsilon \\ \varepsilon & 4 & 4 & \varepsilon \\ 5 & 9 & 9 & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & 0 \end{pmatrix}$$

et la valeur propre associée est 9, le noeud est donc empilé.

La matrice du noeud t_1t_3 est **à compléter**

$$\mathcal{M}(t_1t_3) = \begin{pmatrix} 5 & 9 & 9 & \varepsilon \\ \varepsilon & 4 & 4 & \varepsilon \\ 5 & 9 & 9 & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & 0 \end{pmatrix}$$

et la valeur propre associée est 7, le noeud est également empilé.

La boucle de l'algorithme débute, le noeud t_1t_2 est dépilé et la tâche t_3 est sélectionnée

puisqu'elle n'est pas encore affectée et qu'elle représente le début du travail 2, alors un noeud fils N_{fils} est créé. Le noeud N_{fils} contient les tâches t_1, t_2 et t_3 et sa matrice $\mathcal{M}(N_{fils})$ correspond au produit de la matrice $\mathcal{M}(N)$ par la matrice associée à la tâche t_3 :

$$\mathcal{M}(N_{fils}) = \begin{pmatrix} 7 & 9 & 9 & 7 \\ \varepsilon & 4 & 4 & \varepsilon \\ 7 & 9 & 9 & 7 \\ 2 & \varepsilon & \varepsilon & 2 \end{pmatrix}.$$

Le temps de cycle α en cours est donné par la valeur propre de la matrice $\mathcal{M}(N_{fils})$: $\rho(\mathcal{M}(N_{fils})) = 9$. Le noeud N_{fils} est empilé ensuite la tâche t_4 sélectionnée. La tâche t_4 n'est pas encore affectée et elle succède à t_3 dans le trvail 2, alors un nouveau noeud fils est créé. Il contient les tâches t_1, t_2, t_3 et t_4 et sa matrice est $\mathcal{M}(w_0)$. Ce dernier noeud contient toutes les tâches, un ordonnancement complet est obtenu, cependant la borne supérieur α^+ ne change pas. Il reste le noeud t_1t_3 dans la pile, un noeud fils possible est $t_1t_3t_2$ cependant ce noeud n'est pas considéré car un ordonnancement équivalent a déjà été traité (le noeud $t_1t_2t_3$). Le noeud fils $t_1t_3t_4$ est construit cependant la valeur propre de sa matrice associée est supérieure à la borne supérieure α^+. La procédure est terminée car la pile est vide.

La figure 4.4 affiche l'arbre de recherche pour l'exemple.

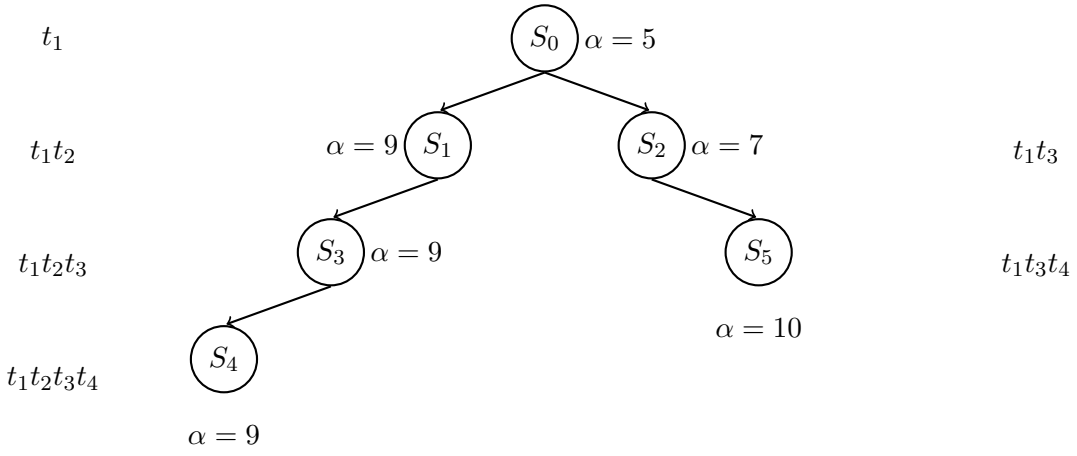

FIGURE 4.4 – Arbre de recherche de la PSE basée sur la théorie des tas pour l'exemple 2.5

4.3 Ordonnancement *K*-cyclique

La théorie des tas telle qu'elle est adaptée aux CJSPs par [GM99] ne prend en considérartion que le cas de CJSP saufs. Dans la première partie de ce chapitre, le modèle présenté est associé aux CJSPs saufs avec des solutions 1-cyclique et un WIP égal à 1. Cette partie est dédiée à la modélisation des ordonnancements K-cycliques. En outre, une généralisation aux CJSPs non saufs est proposée.

4.3.1 Généralisation de la théorie des tas au CJSP non saufs

Le modèle présenté précédemment peut être généralisé pour les ordonnancements K-cyclique. Cependant, ce modèle se limite aux ordonnancements saufs ce qui a deux conséquences sur les ordonnancements concernés par la théorie des tas :

- Il y a au plus une occurrence en activité d'un même travail dans le motif.
- Le motif n doit être terminé avant que le motif $n+1$ débute.

La première caractérisation des ordonnancements concernés signifient que, à l'intérieur d'un motif, les travaux ne s'entrelacent pas. La seconde spécificité est identique au cas 1-cyclique (un seul motif en activité). Pour illustrer ces propos, on prend l'exemple de l'ordonnancement 2-cyclique $w = t_3t_4t_3t_4t_1t_2t_1t_2$ représenté dans la figure 2.7. Cette ordonnancement n'est pas sauf puisqu'à l'intérieur du motif plusieurs occurrences en activité d'un même travail ont lieu (par exemple le travail 2 entre 2 et 7 u.t.) et le motif suivant débute avant la fin du motif courant (deux motifs sont en activité entre 14 et 18 u.t.).

En conservant l'ordre de passage des travaux, l'approche par la théorie des tas permet la modélisation d'un ordonnancement sauf.

Plus précisément, la théorie des tas telle qu'elle est présentée ci-dessus, ne permet de modéliser pour cet ordonnancement $w = t_3t_4t_3t_4t_1t_2t_1t_2$ que la solution 2-cyclique présente dans la figure 4.5. Le tas de pièces correspondant à cette solution est illustrée dans la figure 4.6. On se convainc alors facilement que les ordonnancements K-cycliques saufs ne sont pas les ordonnancements K-cycliques optimaux.

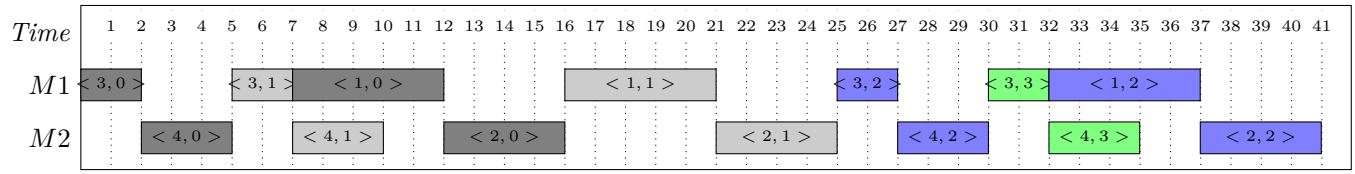

FIGURE 4.5 – Représentation d'une solution de l'exemple de CJSP 2-cyclique associé à un RdP "sauf".

Bien que la complexité de l'évaluation de performances pour un ordonnancement K-cyclique sauf soit identique à celle d'un ordonnancement 1-cyclique sauf (la dimension du tas n'est pas modifiée), la classe des ordonnancements concernés par cette méthode est restreinte et ne concerne que des ordonnacements peu performants.

Nous proposons cependant dans la remarque suivante, une méthode pour considérer des ordonnancements non saufs au prix d'une augmentation de la dimension du tas.

Remarque 4.9. Augmenter la dimension du modèle de type "Tas" associé à un CJSP permet d'élargir l'ensemble des solutions recherchées au cas non "sauf".

L'augmentation envisagée du modèle de type "Tas" correspond à l'ajout de slots associés aux travaux, la dimension du modèle devient alors égale à la somme du nombre de machines plus K fois le nombre de travaux.

Cette extension du tas permet alors de considérer une plus large classe d'ordonnancement K-cyclique. Plus précisément, on peut, dans ces conditions, examiner des ordon-

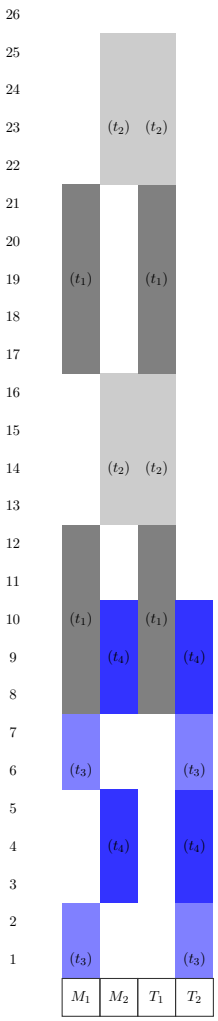

FIGURE 4.6 – Tas de pièces associé à la solution représentée dans la figure 4.5

nancements dont le motif comprend plusieurs occurrences d'un même travail en activité. Cependant, un seul motif ne peut être en activité.

Néanmoins, cette dernière restriction perd de son importance quand le k augmente, c'est-à-dire quand le motif augmente. L'interêt de cette nouvelle méthode permet alors de trouver un équilibre entre complexité raisonnable (étudiée au paragraphe suivant) et performance.

En considérant la remarque 4.9 pour l'exemple 2.5, le tas correspondant à un ordonnancement 2-cyclique est représenté dans la figure 4.7.

La solution associée à au tas de la figure 4.7 est illustrée dans la figure 4.8, cette solution est meilleure que celle représentée dans la figure 4.5, mais elle reste moins bien que la solution 2-cyclique de la figure 2.6.

4.3.2 Complexité du modèle de type "Tas"

Dans un modèle de type "Tas", la dimension du tas, donc des matrices, est égale à $|\mathcal{R}| + |\mathcal{J}|$ la somme du nombre des ressources (machines) et du nombre de travaux dans le

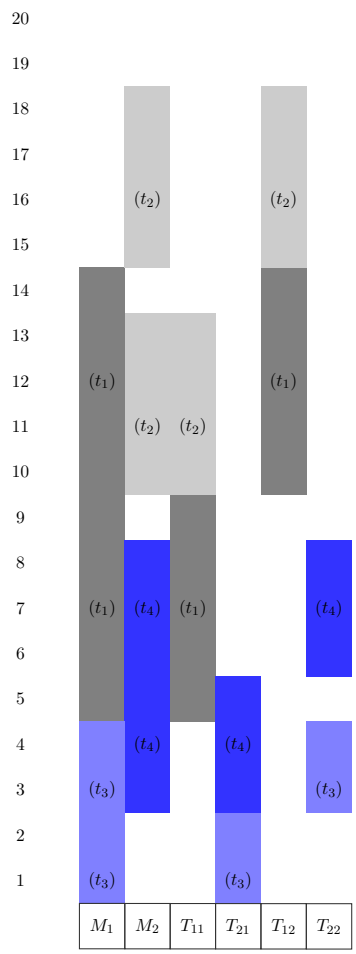

FIGURE 4.7 – Tas de pièces associé à la solution représentée dans la figure 4.8

CJSP associé. Pour calculer le produit de ces matrices la complexité est en $O(|n|(|\mathcal{R}|+|\mathcal{J}|))$ où n est le nombre de tâches du CJSP. La complexité de l'algorithe de Howard n'est pas prouvée, pour avoir une idée de la compléxité totale, on peut considéré l'algorithme de Karp. En utilisant l'algorithme de Karp pour le calcul de valeurs propres, la complexité de ce calcul est en $O((|\mathcal{R}| + |\mathcal{J}|)^3)$. La complexité totale pour résoudre un CJSP avec un modèle "Tas" est $O(|n|(|\mathcal{R}| + |\mathcal{J}|) + (|\mathcal{R}| + |\mathcal{J}|)^3)$.

Il est intéressant de voir l'évolution de cette compléxité dans le cas d'ordonnancement K-cyclique. En ne considérant que les CJSPs "saufs", la taille des matrices reste la même que dans le cas 1-cyclique $|\mathcal{R}| + |\mathcal{J}|$, ce qui change est uniquement le nombre de produits de matrices effectués donc la complexité totale pour une résolution K-cyclique utilisant l'algorithme de Karp est en $O(K|n|(|\mathcal{R}| + |\mathcal{J}|) + (|\mathcal{R}| + |\mathcal{J}|)^3)$.

Dans le cas d'une résolution K-cyclique basée sur d'autre approches, le modèle est K fois plus grand et la compléxité est bien plus importante. L'exemple 4.10 illustre le cas d'une résolution 2-cyclique basée sur la théorie des graphes, la complexité de résolution du CJSP est multipliée par deux dans le cas 2-cyclique.

Exemple 4.10. En prenant l'exemple 2.5, le graphe 4.9 correspond à une cyclicité de 2. Ce graphe est deux fois plus grand que le graphe 2.10. Les arcs associés aux contraintes

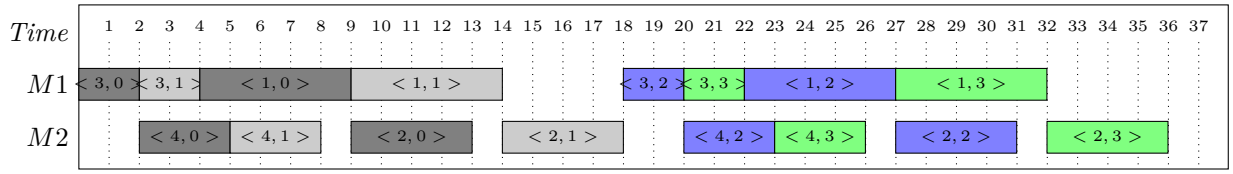

FIGURE 4.8 – Représentation d'une solution de l'exemple de CJSP 2-cyclique avec un WIP=1.

de ressources ne sont pas présents dans le graphe par souci de clarté.

L'approche par la théorie des tas reste avantageuse en terme de complexité, même après l'augmentation de la dimension du modèle générée par adaptation aux CJSPs non "saufs". La compléxité de cette l'approche dans le cas d'ordonnancement K-cyclique adaptée aux CJSPs non "saufs" est certe plus grande que celle du cas 1-cyclique, mais son évolution n'est pas du même ordre que pour les autres approches : La dimension du modèle de type "Tas" devient $|\mathcal{R}| + K|\mathcal{J}|$ et donc la complexité totale pour une résolution K-cyclique utilisant l'algorithme de Karp est en $O(K|n|(|\mathcal{R}| + |\mathcal{J}|) + (|\mathcal{R}| + K|\mathcal{J}|)^3)$.

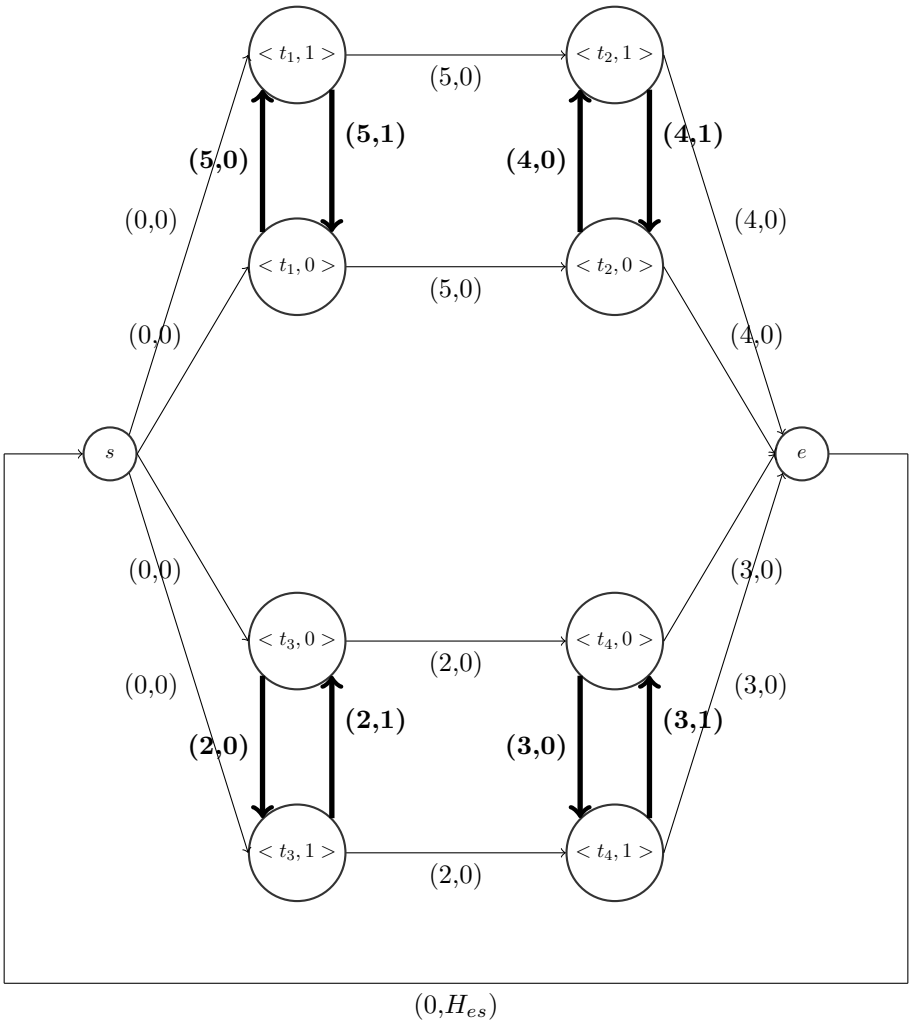

FIGURE 4.9 – Graphe associé de l'exemple de CJSP 2-cyclique

Chapitre 5

Résolution de CJSP - Comparaison des différentes approches

5.1 Comparaison de méthode de résolution d'un CJSP

Dans cette partie, une évaluation des performances de la procédure basée sur la théorie des graphes, est présentée. Dans cet objectif, les deux versions de la nouvelle procédure sont testées en utilisant des instances de CJSP générées aléatoirement. Les résultats sont ensuite comparés avec les performances de la procédure de Hanen et du programme linéaire mixte (MLP) décrit dans le paragraphe 2.2.4. Cette étude comparative a fait aussi l'objet d'une publication (voir [BHF12]).

5.1.1 Comparaison des différentes procédures de branch and bound

Les quatre implémentations de procédures de branch and bound (BnB) pour résoudre le CJSP sont comparées : la nouvelle procédure version 1, la nouvelle procédure version 2, la procédure de Hanen avec l'algorithme de Howard et la procédure de Hanen avec l'algorithme de Gondran et Minoux. Les différentes caractéristiques sont résumées dans le tableau 5.1.

En plus des caractéristiques citées dans le tableau 5.1, l'algorithme de la nouvelle procédure fait un branchement en profondeur d'abord et calcule ensuite le temps de cycle pour tous les noeuds. L'algorithme de Hanen fait aussi un branchement en profondeur d'abord mais le circuit critique est calculé uniquement quand il trouve une solution complète.

5.1.2 Caractéristiques techniques

Les tests des quatre implantations sont effectués sur un ordinateur avec un AMD Athlon 64 x2 3800+ processor, 1.7 GB RAM et avec Linux Fedora 9 comme système d'exploitation. Les instances de CJSP sont générées aléatoirement. Elles sont caractérisées par un nombre de travaux, un nombre de tâches et un nombre de machines. Six différents

TABLE 5.1 – Les différentes caractéristiques des procédures de BnB pour résoudre le CJSP

Procédure	La notre		Hanen	
Version	Version 1	Version 2	Version 1	Version 2
Algorithme pour le GBCSP	Howard	Howard	Howard	Gondran et Minoux
Calcul des bornes	Consistance du graphe, m.à.j à chaque noeud	Consistance du graphe, vérifié avec Howard	Amplitude	Amplitude
Rejet des noeuds	$\alpha \geq \alpha^+$	$\alpha \geq \alpha^+$, noeud infaisable	Circuit d'amplitude positive	Circuit d'amplitude positive
Fin de la procédure	Pile de noeuds vide, $\alpha^- = \alpha^+$	Pile de noeuds vide, $\alpha^- = \alpha^+$	Pile de noeuds vide, $\alpha^- = \alpha^+$	Pile de noeuds vide, $\alpha^- = \alpha^+$
Sélection de noeud	Plus petit α	Plus petit α	Basé sur les amplitudes de circuit	Basé sur les amplitudes de circuit
Règle de branchement	K_{ij} dans le circuit critique, Plus petit intervalle	Plus petit intervalle	Basé sur les amplitudes de circuit	Basé sur les amplitudes de circuit

types d'instances de CJSP sont testés, chacun est composé de dix problèmes, qu'on peut voir dans le tableau 5.2.

TABLE 5.2 – Aperçu sur les types de problèmes

Type	S_1	S_2	M_1	M_2	L_1	L_2
Nombre de travaux	8	5	5	8	5	5
Nombre de tâches	50	50	50	80	100	100
Nombre de machines	4	10	5	4	10	5

Pour chacun des six types de problèmes, dix instances ont été résolues. La durée d'exécution de chaque implémentation est limitée à 180 secondes.

5.2 Résolution de problèmes d'ordonnancement 1-cyclique avec un WIP = K

5.2.1 Résultats numériques

Les résultats numériques des tests pour les cinq implémentations (les quatre PSE et le MLP) sont présentés dans cette partie. Les solutions considérées sont 1-cyclique avec un WIP égal à 2 ou 3. Dans le tableau 5.3 (resp. le tableau 5.4) sont notés les nombres de problèmes résolus dans un intervalle de temps de trois minute par chacune des méthodes quand le WIP est égal à 2 (resp. 3).

TABLE 5.3 – Résultats numériques – Nombre d'instances résolues avec $WIP = 2$

Type	S_1	S_2	M_1	M_2	L_1	L_2
Notre procédure version 1	10	10	8	2	0	0
Notre procédure version 2	4	4	3	1	0	0
MILP	9	10	7	0	0	0
Procédure de Hanen Howard	10	9	7	2	0	0
Procédure de Hanen Gondran et Minoux	0	6	0	0	0	0

TABLE 5.4 – Résultats numériques – Nombre d'instances résolues avec $WIP = 3$

Type	S_1	S_2	M_1	M_2	L_1	L_2
Notre procédure version 1	9	8	9	8	1	3
Notre procédure version 2	4	3	3	3	0	0
MILP	10	10	9	0	0	0
Procédure de Hanen Howard	9	7	7	8	0	3
Procédure de Hanen Gondran et Minoux	0	0	0	0	0	0

Les tableaux 5.5 et 5.6 rapportent des résultats plus détaillés pour des solutions avec un WIP égal à 2.

Les tableaux 5.7 et 5.8 illustrent les résultats détaillés pour des solutions avec un WIP égal à 3.

5.2.2 Interprétation des résultats numériques

Les résultats de tests sont interprétés en deux temps : tout d'abord, nous analysons les différentes performances des deux versions de la nouvelle procédure. Les deux versions diffèrent considérablement. Bien évidemment, la mise à jour de la matrice des hauteurs prend plus de temps que le fait de laisser l'algorithme de Howard examiner la faisabilité d'un noeud. Mais, cette mise à jour permet d'élager l'arbre de recherche, ce qui diminue largement le nombre de noeuds à examiner dans la version 1, comme les bornes inférieures des décalages événementiels ne peuvent qu'augmenter au cours de la procédure. Le nombre d'instances résolues par la nouvelle procédure version 2 diminue puisque l'algorithme est gourmand en terme de capacité de mémoire, et pas à cause du temps d'exécution (ceci est reporté dans les tableaux de résultats par une étoile.).

Pour la procédure de Hanen, les résultats de la procédure avec l'algorithme de Gondran et Minoux sont mis en annexe. Même quand cette version arrive à trouver un optimum, elle ne garantit pas l'optimalité dans la quasi-totalité des cas, contrairement à la version de la procédure de Hanen avec l'algorithme de Howard. Ce qui s'explique par la compléxité de l'algorithme de Gondran et Minoux qui est en $O(n^3 log(n))$ alors que l'algorithme de Howard nécessite un temps de calcul presque linéaire.

Ensuite, pour la nouvelle procédure, seule la version 1 est considérée par la suite. Toutes les implémentations sont comparées les unes aux autres, et dans ce cas, trois genres de problèmes sont différenciés : les problèmes de petite taille (type S_1 et S_2), les problèmes

TABLE 5.5 – Résultats numériques avec $H_{es} = 2$

	$H_{es} = 2$							
	Nouvelle Méthode V1		Nouvelle Méthode V2		Hanen		MILP	
	Solution	Temps	Solution	Temps	Solution	Temps	Solution	Temps
cjsp_8_50_4_#1	**99**	3.45	**99**	3.59	**99**	0.04	**99**	3.97
cjsp_8_50_4_#2	**83**	0.35	*	*	**83**	0.1	**83**	114.91
cjsp_8_50_4_#3	**91**	4.56	*	*	**91**	0.6	**91**	12.86
cjsp_8_50_4_#4	**92**	7.05	*	*	**92**	0.04	**92**	17.9
cjsp_8_50_4_#5	**81**	16.41	**81**	17.13	**81**	0.04	**81**	164.08
cjsp_8_50_4_#6	**104**	2.26	**104**	2.37	**104**	0.05	**104**	4.03
cjsp_8_50_4_#7	**87**	38.77	**87**	40.53	**87**	0.04	**87**	0.62
cjsp_8_50_4_#8	**80**	12.28	*	*	**80**	0.05	**80**	0.23
cjsp_8_50_4_#9	**99**	52.82	*	*	**99**	0.04	**99**	4.48
cjsp_8_50_4_#10	**73**	0.09	*	*	**73**	0.05	76	179.95
cjsp_5_50_10_#1	**49**	0.21	**49**	0.51	**49**	4.99	**49**	0.75
cjsp_5_50_10_#2	**49**	0.15	**49**	0.49	**49**	0.14	**49**	0.51
cjsp_5_50_10_#3	**49**	60.43	*	*	**49**	60.43	**49**	1.99
cjsp_5_50_10_#4	**42**	11.68	*	*	**42**	11.68	**42**	2.02
cjsp_5_50_10_#5	**54**	0.73	**54**	1.43	**54**	7.81	**54**	2.98
cjsp_5_50_10_#6	**53**	0.06	*	*	**53**	0.06	**53**	1.89
cjsp_5_50_10_#7	**59**	0.14	*	*	**59**	0.14	**59**	0.81
cjsp_5_50_10_#8	**48**	33.6	**48**	54.5	59	180.01	**48**	7.62
cjsp_5_50_10_#9	**48**	15.33	*	*	**48**	15.33	**48**	2.49
cjsp_5_50_10_#10	**52**	0.08	*	*	**52**	17.18	**52**	2.32
cjsp_5_50_5_#1	**65**	27.25	*	*	**65**	30.63	**65**	5.45
cjsp_5_50_5_#2	**70**	14.71	*	*	**70**	16.57	71	179.94
cjsp_5_50_5_#3	85	180.01	*	*	87	180.01	**83**	11.21
cjsp_5_50_5_#4	**83**	2.34	**83**	10.49	**83**	2.66	**83**	72.49
cjsp_5_50_5_#5	**87**	0.03	*	*	**87**	0.03	**87**	102.73
cjsp_5_50_5_#6	**79**	2.66	**79**	1.87	85	180.01	**79**	4.07
cjsp_5_50_5_#7	80	180.01	*	*	80	180.01	78	179.93
cjsp_5_50_5_#8	**72**	1.79	**72**	14.2	**72**	1.57	**72**	2.97
cjsp_5_50_5_#9	**78**	0.25	*	*	**78**	0.25	80	179.94
cjsp_5_50_5_#10	**85**	0.04	*	*	**85**	0.04	**85**	43.93
cjsp_8_80_4_#1	143	180.01	*	*	147	180.01	144	179.93
cjsp_8_80_4_#2	125	180.01	*	*	128	180.01	144	179.94
cjsp_8_80_4_#3	152	180.01	*	*	155	180.01	166	179.93
cjsp_8_80_4_#4	134	180.01	*	*	134	180.01	154	179.94
cjsp_8_80_4_#5	**145**	27.05	*	*	**145**	28.33	168	179.93
cjsp_8_80_4_#6	130	180.01	*	*	130	180.01	139	179.94
cjsp_8_80_4_#7	**112**	1.97	*	*	**112**	2.07	125	179.93
cjsp_8_80_4_#8	128	180.01	*	*	128	180.01	148	179.93
cjsp_8_80_4_#9	156	180.01	191	180.01	181	180.01	157	179.94
cjsp_8_80_4_#10	146	180.01	*	*	143	180.01	152	179.93

de taille moyenne (type M_1 et M_2) et les problèmes de grande taille (type L_1 et L_2).

Pour les problèmes de petite taille (type S_1, S_2), en considérant les deux valeurs de WIP, les trois procédures se comportent bien et résolvent quasiment les 10 problèmes en moins de trois minutes. Donc, pour ce genre de problèmes, aucune différence significative n'est observée entre les différentes procédures. A noter que le MLP est particulièrement plus efficace sur les instances `cjsp_5_50_10` que sur les `cjsp_8_50_4`.

Dans le cas des CJSPs de taille moyenne (type M_1, M_2), la nouvelle procédure résout 10 (resp. 17) instances avec un WIP $H_{es} = 2$ (resp. $H_{es} = 3$), quand la procédure de Hanen résout 9 (resp. 15) instances avec un WIP $H_{es} = 2$ (resp. $H_{es} = 3$). Le MLP marche un

TABLE 5.6 – Résultats numériques avec $H_{es} = 2$ (Suite)

	$H_{es} = 2$							
	Nouvelle Méthode V1		Nouvelle Méthode V2		Hanen		MILP	
	Solution	Temps	Solution	Temps	Solution	Temps	Solution	Temps
cjsp_5_100_10_#1	130	180.01	115	180.01	130	180.01	107	179.95
cjsp_5_100_10_#2	114	180.01	112	180.01	114	180.01	117	179.95
cjsp_5_100_10_#3	105	180.01	148	180.01	146	180.01	111	179.95
cjsp_5_100_10_#4	116	180.01	140	180.01	116	180.01	109	179.95
cjsp_5_100_10_#5	86	180.01	108	180.01	96	180.01	91	179.95
cjsp_5_100_10_#6	110	180.01	121	180.01	114	180.01	115	179.96
cjsp_5_100_10_#7	103	180.01	113	180.01	110	180.01	105	179.95
cjsp_5_100_10_#8	102	180.01	115	180.01	114	180.01	97	179.95
cjsp_5_100_10_#9	91	180.01	112	180.01	100	180.01	82.5	179.95
cjsp_5_100_10_#10	106	180.01	115.5	180.01	111	180.01	96	179.95
cjsp_5_100_5_#1	166	180.01	204	180.01	166	180.01	177	179.94
cjsp_5_100_5_#2	147	180.01	170	180.01	147	180.01	155	179.94
cjsp_5_100_5_#3	148	180.01	217	180.01	145	180.01	169	179.94
cjsp_5_100_5_#4	162	180.01	197	180.01	162	180.01	181	179.93
cjsp_5_100_5_#5	135	180.01	162	180.01	142	180.01	147	179.94
cjsp_5_100_5_#6	143	180.01	178	180.01	143	180.01	150	179.94
cjsp_5_100_5_#7	166	180.01	201	180.01	163	180.01	174	179.94
cjsp_5_100_5_#8	136	180.01	173	180.01	140	180.01	147	179.94
cjsp_5_100_5_#9	150	180.01	211	180.01	153	180.01	158	179.92
cjsp_5_100_5_#10	165	180.01	189	180.01	168	180.01	174	179.93

moins bien pour ce genre de CJSP car il ne résout qu'une seule instance des M_2 en moins de trois minutes.

Pour CJSPs de grande taille (type L_1, L_2), aucune des procédures ne résout d'instances en moins de trois minutes pour un WIP $H_{es} = 2$. On remarque cependant, que la procédure de Hanen n'est pas dominante par rapport à la nouvelle procédure pour les CJSPs de type M_1, tandis que le MLP n'est pas non plus dominant pour les instances M_2. Quand la valeur du WIP est augmentée à 3, le MLP n'est plus en mesure de résoudre les différentes instances en moins de trois minutes, la nouvelle procédure en résout 5 et la procédure de Hanen 4.

5.3 Résolution de problèmes d'ordonnancement K-cyclique avec un WIP = 1

Dans cette partie, les solutions recherchées sont 2-cyclique ou 3-cyclique avec un WIP égal à 1. Ces ordonnancements sont plus adaptés à l'approche par la théorie des tas, ce qui explique les mauvais résultats obtenus par les différentes méthodes considérées : la nouvelle PSE version 1, la PSE de Hanen avec l'algorithme de Howard et le MLP.

5.3.1 Résultats numériques

Le tableau 5.9 affiche les résultats des trois méthodes avec des solutions 2-cycliques. Les types de CJSP testés ici sont S_1, S_2 et M_1. En tenant compte de la définition des ordonnancements K-cycliques, le temps de cycle se calcule selon l'équation (2.12). Pour

TABLE 5.7 – Résultats Numériques avec $H_{es} = 3$

	$H_{es} = 3$							
	Nouvelle Méthode V1		Nouvelle Méthode V2		Hanen		MILP	
	Solution	Temps	Solution	Temps	Solution	Temps	Solution	Temps
cjsp_8_50_4_#1	**99**	0.05	**99**	9.06	**99**	0.06	**99**	63.76
cjsp_8_50_4_#2	**83**	0.06	**83**	4.87	**83**	0.07	**83**	12.28
cjsp_8_50_4_#3	96	180.01	*	*	96	180.01	**91**	9.6
cjsp_8_50_4_#4	**92**	0.05	*	*	**92**	0.05	**92**	94.51
cjsp_8_50_4_#5	**81**	0.05	**81**	42.81	**81**	0.05	**81**	3.91
cjsp_8_50_4_#6	**104**	0.05	**104**	2.33	**104**	0.05	**104**	3.38
cjsp_8_50_4_#7	**87**	7.11	*	*	**87**	8.04	**87**	0.32
cjsp_8_50_4_#8	**80**	0.04	*	*	**80**	0.06	**80**	6.88
cjsp_8_50_4_#9	**99**	0.05	*	*	**99**	0.05	**99**	3.36
cjsp_8_50_4_#10	**73**	0.06	*	*	**73**	0.05	**73**	168.19
cjsp_5_50_10_#1	51	180.01	*	*	51	180.01	**40**	0.99
cjsp_5_50_10_#2	**49**	0.1	**49**	24.94	**49**	0.11	**49**	1.98
cjsp_5_50_10_#3	**44**	8.29	*	*	**44**	8.61	**44**	1.38
cjsp_5_50_10_#4	42	180.01	*	*	45	180.01	**39**	2.11
cjsp_5_50_10_#5	**54**	0.02	*	*	**54**	0.03	**54**	0.35
cjsp_5_50_10_#6	**53**	0.17	*	*	**53**	0.19	**53**	1.13
cjsp_5_50_10_#7	**59**	0.03	*	*	**59**	0.02	**59**	1.74
cjsp_5_50_10_#8	**45**	0.97	**45**	86.8	47	180.01	**45**	1.11
cjsp_5_50_10_#9	**45**	0.37	*	*	**45**	0.4	**45**	0.9
cjsp_5_50_10_#10	**52**	1.9	**52**	74.8	**52**	1.9	**52**	1.84
cjsp_5_50_5_#1	67	180.01	*	*	**65**	30.74	**65**	164.97
cjsp_5_50_5_#2	**70**	21.07	*	*	**70**	16.65	**70**	15.71
cjsp_5_50_5_#3	**83**	0.04	*	*	87	180.01	**83**	96.81
cjsp_5_50_5_#4	**83**	0.42	**83**	26.29	**83**	2.67	**83**	14.2
cjsp_5_50_5_#5	**87**	0.04	*	*	**87**	0.04	**87**	80.7
cjsp_5_50_5_#6	**79**	0.34	**79**	2.4	85	180.01	**79**	12.21
cjsp_5_50_5_#7	**77**	0.04	*	*	80	180.01	**77**	6.95
cjsp_5_50_5_#8	**72**	11.6	**72**	21.86	**72**	1.77	**72**	4.04
cjsp_5_50_5_#9	**78**	0.41	*	*	**78**	0.25	79	179.95
cjsp_5_50_5_#10	**85**	2.12	*	*	**85**	0.04	**85**	64.92
cjsp_8_80_4_#1	**141**	0.38	*	*	**141**	0.35	161	179.94
cjsp_8_80_4_#2	**120**	3.36	*	*	**120**	0.42	131	179.94
cjsp_8_80_4_#3	155	180.01	200	180.01	155	180.01	155	179.93
cjsp_8_80_4_#4	**129**	0.16	*	*	**129**	0.47	144	179.94
cjsp_8_80_4_#5	**145**	0.44	*	*	**145**	0.44	154	179.93
cjsp_8_80_4_#6	**121**	0.28	153	180.01	**121**	0.42	142	179.93
cjsp_8_80_4_#7	115	180.01	*	*	115	180.01	134	179.93
cjsp_8_80_4_#8	**124**	1.8	*	*	**124**	0.84	125	179.94
cjsp_8_80_4_#9	**144**	0.42	191	180.01	**144**	0.43	154	179.94
cjsp_8_80_4_#10	**133**	0.32	*	*	**133**	0.4	**133**	174.37

ces résultats la solution obtenue est à diviser par deux pour obtenir le temps de cycle.

Le tableau 5.10 illustre les résultats des trois méthodes considérées précédemment avec des solutions 3-cycliques. Le temps de cycle est obtenu en divisant les solutions obtenues par trois.

5.3.2 Interprétation des résultats numériques

Au regard des résultats numériques obtenus dans cette partie, l'utilisation des PSEs basées sur la théorie des graphes et du MLP n'est appropriée pour rechercher de solutions

TABLE 5.8 – Résultats Numériques avec $H_{es} = 3$ (Suite)

	$H_{es} = 3$							
	Nouvelle Méthode V1		Nouvelle Méthode V2		Hanen		MILP	
	Solution	Temps	Solution	Temps	Solution	Temps	Solution	Temps
cjsp_5_100_10_#1	122	180.01	125	180.01	130	180.01	100	179.95
cjsp_5_100_10_#2	102	180.01	132	180.01	114	180.01	99	179.95
cjsp_5_100_10_#3	138	180.01	121	180.01	146	180.01	107	179.95
cjsp_5_100_10_#4	105	180.01	145	180.01	116	180.01	100	179.95
cjsp_5_100_10_#5	93	180.01	102	180.01	96	180.01	90	179.95
cjsp_5_100_10_#6	**108**	0.68	124	180.01	114	180.01	111	179.95
cjsp_5_100_10_#7	112	180.01	113	180.01	110	180.01	107	179.95
cjsp_5_100_10_#8	103	180.01	125	180.01	114	180.01	95	179.95
cjsp_5_100_10_#9	95	180.01	106	180.01	100	180.01	79	179.95
cjsp_5_100_10_#10	111	180.01	115.5	180.01	111	180.01	95	179.94
cjsp_5_100_5_#1	**145**	1.74	204	180.01	**145**	2.11	153	179.94
cjsp_5_100_5_#2	152	180.01	170	180.01	152	180.01	168	179.93
cjsp_5_100_5_#3	**143**	105.97	217	180.01	**143**	106.01	158	179.94
cjsp_5_100_5_#4	**155**	35.71	197	180.01	**155**	67.37	177	179.94
cjsp_5_100_5_#5	126	180.01	162	180.01	138	180.01	159	179.93
cjsp_5_100_5_#6	144	180.01	178	180.01	144	180.01	159	179.94
cjsp_5_100_5_#7	162	180.01	201	180.01	179	180.01	166	179.94
cjsp_5_100_5_#8	151	180.01	173	180.01	151	180.01	155	179.93
cjsp_5_100_5_#9	142	180.01	211	180.01	142	180.01	152	179.94
cjsp_5_100_5_#10	**154**	0.99	189	180.01	**154**	0.68	189	179.93

K-cycliques. En effet, en moins de trois minutes, seules quatre instances du type S_1 sont résolues par la nouvelle PSE version 1 et trois instances du même type résolues par la PSE de Hanen avec l'algorithme de Howard dans le cas 2-cyclique. Toutes les autres instances restent non résolues. Ces derniers résultats montrent l'interêt d'une approche plus simplifiée pour résoudre le problème K-cyclique.

TABLE 5.9 – Résultats Numériques avec des solutions 2-cycliques

	Solutions 2-cycliques					
	Nouvelle Méthode V1		Hanen V2		MILP	
	Solution	Temps	Solution	Temps	Solution	Temps
cjsp_8_50_4_#1	**198**	0.38	**198**	0.65	221	179.39
cjsp_8_50_4_#2	173	180.01	173	180.01	220	179.37
cjsp_8_50_4_#3	218	180.01	220	180.01	210	179.38
cjsp_8_50_4_#4	192	180.01	193	180.01	222	179.37
cjsp_8_50_4_#5	165	180.01	165	180.01	167	179.38
cjsp_8_50_4_#6	**208**	0.05	**208**	0.89	220	179.39
cjsp_8_50_4_#7	187	180.01	199	180.01	215	179.38
cjsp_8_50_4_#8	**160**	80.1	169	180.01	197	179.39
cjsp_8_50_4_#9	**198**	0.65	**198**	0.92	234	179.38
cjsp_8_50_4_#10	198	180.01	198	180.01	190	179.37
cjsp_5_50_10_#1	139	180.01	139	180.01	110	179.44
cjsp_5_50_10_#2	110	180.01	115	180.01	115	179.44
cjsp_5_50_10_#3	113	180.01	113	180.01	113	179.43
cjsp_5_50_10_#4	125	180.01	126	180.01	121	179.42
cjsp_5_50_10_#5	134	180.01	128	180.01	120	179.42
cjsp_5_50_10_#6	120	180.01	145	180.01	138	179.42
cjsp_5_50_10_#7	139	180.01	139	180.01	130	179.42
cjsp_5_50_10_#8	100	180.01	135	180.01	121	179.42
cjsp_5_50_10_#9	127	180.01	137	180.01	127	179.42
cjsp_5_50_10_#10	109	180.01	132	180.01	117	179.43
cjsp_5_50_5_#1	105	180.01	134	180.01	143	179.42
cjsp_5_50_5_#2	177	180.01	188	180.01	177	179.43
cjsp_5_50_5_#3	173	180.01	173	180.01	194	179.45
cjsp_5_50_5_#4	183	180.01	175	180.01	186	179.38
cjsp_5_50_5_#5	187	180.01	204	180.01	209	179.38
cjsp_5_50_5_#6	179	180.01	171	180.01	197	179.38
cjsp_5_50_5_#7	166	180.01	166	180.01	168	179.38
cjsp_5_50_5_#8	152	180.01	155	180.01	189	179.38
cjsp_5_50_5_#9	186	180.01	189	180.01	225	179.38
cjsp_5_50_5_#10	198	180.01	213	180.01	207	179.43

TABLE 5.10 – Résultats Numériques avec des solutions 3-cycliques

	Solutions 3-cycliques					
	Nouvelle Méthode V1		Hanen V2		MILP	
	Solution	Temps	Solution	Temps	Solution	Temps
cjsp_8_50_4_#1	398	180.01	398	180.01	398	179.39
cjsp_8_50_4_#2	273	180.01	273	180.01	320	179.42
cjsp_8_50_4_#3	296	180.01	320	180.01	313	179.39
cjsp_8_50_4_#4	353	180.01	353	180.01	324	179.38
cjsp_8_50_4_#5	256	180.01	265	180.01	268	179.38
cjsp_8_50_4_#6	378	180.01	378	180.01	410	179.37
cjsp_8_50_4_#7	370	180.01	384	180.01	370	179.38
cjsp_8_50_4_#8	269	180.01	269	180.01	291	179.43
cjsp_8_50_4_#9	462	180.01	434	180.01	417	179.42
cjsp_8_50_4_#10	227	180.01	227	180.01	298	179.37
cjsp_5_50_10_#1	151	180.01	159	180.01	150	179.39
cjsp_5_50_10_#2	167	180.01	170	180.01	167	179.44
cjsp_5_50_10_#3	150	180.02	179	180.01	150	179.45
cjsp_5_50_10_#4	230	180.01	146	180.01	146	179.41
cjsp_5_50_10_#5	302	180.01	299	180.01	302	179.42
cjsp_5_50_10_#6	198	180.01	220	180.01	237	179.41
cjsp_5_50_10_#7	196	180.01	192	180.01	196	179.41
cjsp_5_50_10_#8	150	180.01	150	180.01	150	179.41
cjsp_5_50_10_#9	152	180.01	166	180.01	172	179.43
cjsp_5_50_10_#10	156	180.01	159	180.02	161	179.43
cjsp_5_50_5_#1	170	180.01	164	180.01	151	179.41
cjsp_5_50_5_#2	270	180.01	288	180.01	277	179.45
cjsp_5_50_5_#3	319	180.01	323	180.01	319	179.41
cjsp_5_50_5_#4	273	180.01	257	180.02	276	179.39
cjsp_5_50_5_#5	261	180.01	261	180.01	304	179.38
cjsp_5_50_5_#6	258	180.01	271	180.01	279	179.39
cjsp_5_50_5_#7	337	180.01	362	180.01	351	179.39
cjsp_5_50_5_#8	220	180.02	219	180.01	220	179.39
cjsp_5_50_5_#9	258	180.01	264	180.01	261	179.38
cjsp_5_50_5_#10	325	180.01	331	180.01	325	179.43

Conclusion et perspectives

Ce travail de thèse est dédié à la résolution de problème de Job-Shop cyclique.

Nous avons proposé une nouvelle procédure pour résoudre le CJSP. Ses deux principales caractéristiques sont l'évaluation des noeuds par l'algorithme de Howard et le calcul de bornes basées sur la consistance du graphe générique. Nous avons implémenté deux versions de cette nouvelle procédure qui utilisent différentes approches de délimitation de l'arbre de recherche. Puis, nous les avons comparés à trois autres implémentations de procédures pour la résolution de CJSP : La PSE de Hanen avec l'algorithme de Howard, la PSE de Hanen avec l'algorithme de Gondran et Minoux et un MLP résolu avec CPLEX. Les tests ont démontré que pour les problèmes de petite taille, les trois méthodes se valent. Pour les problèmes plus grande taille, notre procédure est compétitive voire plus performante que les deux autres.

Bien que les résultats soient prometteurs, la nouvelle procédure a encore un potentiel d'amélioration et devrait être soumise à de futures recherches pour obtenir des résultats encore meilleurs, par exemple, l'amélioration des fonctions de sélection des noeuds et de la borne inférieure du temps de cycle.

En plus de l'amélioration de la procédure elle-même, d'autres expériences de calcul pourraient être effectuées, comme des tests sur des instances qui génèrent un très grand nombre d'arcs disjonctifs. Pour être en mesure de faire face à ce genre de problèmes, une présélection d'arcs disjonctifs en utilisant une heuristique semble être une solution encourageante.

Les trois méthodes ont été testées pour différents types de CJSP mais aussi différents types de solutions, dans un premier temps, le choix était de fixer la cyclicité à 1 (ordonnancements 1-cycliques) et de faire varier le work-in-process (2 ou 3). Ensuite, c'est la valeur du work-in-process qui est fixée à 1 et la cyclicité varie entre 2 et 3 (ordonnancements 2-cycliques ou 3-cycliques). Ce deuxième type de solutions peut être très intéressant si on considère la procédure basée la nouvelle approche présentée : l'approche par la théorie des tas.

Cette approche originale de la théorie des tas pour résoudre les problèmes d'ordonnancement cyclique ne concerne que les ordonnancements K-cycliques, avec un K fixé, associés à des ordonnancement saufs. Un élargissement de l'ensemble des solutions est proposé mais avec une augmentation du nombre de slots, donc de la complexité du modèle. Cet extension du tas permet de considérer une plus large classe d'ordonnancement cyclique (sans toutefois en considérer la totalité). Cette perspective se révele être prometteuse car la complexité , même si elle est plus importante que pour le cas 1-périodique,

reste raisonnable et on peut espérer de bonnes performances de ces ordonnancements (les ordonnancements K-cycliques sont dominants). Il reste à mettre en oeuvre cette comparaison, elle fera l'objet de prochaines études. La mise en oeuvre de l'extension de la méthode de la théorie des tas pourra donner lieu à une comparaison entre des approches 1-cycliques avec un WIP supérieur à 1 et des approches K-cycliques avec un WIP égal à 1. Les méthodes actuelles, basées sur des graphes ou sur la MLP, se révèlent trop inéfficaces dans le cas K-cyclique pour autoriser cette comparaison.

En outre, nous pouvons citer comme perspective de ce travail, la recherche d'une borne supérieure du WIP afin qu'une solution 1-cyclique soit optimale aussi bien par rapport à la minimisation du temps du cycle que par rapport à la minimisation du WIP. Cette question se pose également pour les ordonnancements K-cycliques.

Annexes

Annexes A

TABLE 5.11 – Résultats numériques pour la procédure de Hanen avec $H_{es} = 2$

	$H_{es} = 2$			
	Hanen V2		Hanen V1	
	Solution	Temps	Solution	Temps
cjsp_8_50_4_#1	**99**	0.04	99	180.01
cjsp_8_50_4_#2	**83**	0.1	83	180.01
cjsp_8_50_4_#3	**91**	0.6	91	180.01
cjsp_8_50_4_#4	**92**	0.04	92	180.01
cjsp_8_50_4_#5	**81**	0.04	81	180.01
cjsp_8_50_4_#6	**104**	0.05	104	180.01
cjsp_8_50_4_#7	**87**	0.04	87	180.01
cjsp_8_50_4_#8	**80**	0.05	80	180.01
cjsp_8_50_4_#9	**99**	0.04	99	180.01
cjsp_8_50_4_#10	**73**	0.05	73	180.01
cjsp_5_50_10_#1	**49**	4.99	**49**	4.49
cjsp_5_50_10_#2	**49**	0.14	**49**	13.18
cjsp_5_50_10_#3	**49**	60.43	**49**	50.9
cjsp_5_50_10_#4	**42**	11.68	**42**	9.09
cjsp_5_50_10_#5	**54**	7.81	54	180.01
cjsp_5_50_10_#6	**53**	0.06	**53**	72.7
cjsp_5_50_10_#7	**59**	0.14	59	180.01
cjsp_5_50_10_#8	59	180.01	59	180.01
cjsp_5_50_10_#9	**48**	15.33	**48**	12.79
cjsp_5_50_10_#10	**52**	17.18	52	180.01
cjsp_5_50_5_#1	**65**	30.63	65	180.01
cjsp_5_50_5_#2	**70**	16.57	70	180.01
cjsp_5_50_5_#3	87	180.01	83	180.01
cjsp_5_50_5_#4	**83**	2.66	83	180.01
cjsp_5_50_5_#5	**87**	0.03	87	180.01
cjsp_5_50_5_#6	85	180.01	85	180.01
cjsp_5_50_5_#7	80	180.01	80	180.01
cjsp_5_50_5_#8	**72**	1.57	72	180.01
cjsp_5_50_5_#9	**78**	0.25	78	180.01
cjsp_5_50_5_#10	**85**	0.04	85	180.01

Annexes B

TABLE 5.12 – Résultats numériques pour la procédure de Hanen avec $H_{es} = 2$ (Suite)

	$H_{es} = 2$			
	Hanen V2		Hanen V1	
	Solution	Temps	Solution	Temps
cjsp_8_80_4_#1	147	180.01	147	180.01
cjsp_8_80_4_#2	128	180.01	128	180.01
cjsp_8_80_4_#3	155	180.01	155	180.01
cjsp_8_80_4_#4	134	180.01	134	180.01
cjsp_8_80_4_#5	**145**	28.33	145	180.01
cjsp_8_80_4_#6	130	180.01	130	180.01
cjsp_8_80_4_#7	**112**	2.07	112	180.01
cjsp_8_80_4_#8	128	180.01	128	180.01
cjsp_8_80_4_#9	181	180.01	181	180.01
cjsp_8_80_4_#10	143	180.01	143	180.01
cjsp_5_100_10_#1	130	180.01	130	180.01
cjsp_5_100_10_#2	114	180.01	114	180.01
cjsp_5_100_10_#3	146	180.01	146	180.01
cjsp_5_100_10_#4	116	180.01	116	180.01
cjsp_5_100_10_#5	96	180.01	96	180.01
cjsp_5_100_10_#6	114	180.01	114	180.01
cjsp_5_100_10_#7	110	180.01	110	180.01
cjsp_5_100_10_#8	114	180.01	114	180.01
cjsp_5_100_10_#9	100	180.01	100	180.01
cjsp_5_100_10_#10	111	180.01	111	180.01
cjsp_5_100_5_#1	166	180.01	166	180.01
cjsp_5_100_5_#2	147	180.01	147	180.01
cjsp_5_100_5_#3	145	180.01	145	180.01
cjsp_5_100_5_#4	162	180.01	162	180.01
cjsp_5_100_5_#5	142	180.01	142	180.01
cjsp_5_100_5_#6	143	180.01	143	180.01
cjsp_5_100_5_#7	163	180.01	163	180.01
cjsp_5_100_5_#8	140	180.01	140	180.01
cjsp_5_100_5_#9	153	180.01	153	180.01
cjsp_5_100_5_#10	168	180.01	168	180.01

Annexes C

TABLE 5.13 – Résultats numériques pour la procédure de Hanen avec $H_{es} = 3$

	$H_{es} = 3$			
	Hanen V2		Hanen V1	
	Solution	Temps	Solution	Temps
cjsp_8_50_4_#1	**99**	0.06	99	180.01
cjsp_8_50_4_#2	**83**	0.07	83	180.01
cjsp_8_50_4_#3	96	180.01	96	180.01
cjsp_8_50_4_#4	**92**	0.05	92	180.01
cjsp_8_50_4_#5	**81**	0.05	81	180.01
cjsp_8_50_4_#6	**104**	0.05	104	180.01
cjsp_8_50_4_#7	**87**	8.04	87	180.01
cjsp_8_50_4_#8	**80**	0.06	80	180.02
cjsp_8_50_4_#9	**99**	0.05	99	180.02
cjsp_8_50_4_#10	**73**	0.05	73	180.01
cjsp_5_50_10_#1	51	180.01	51	180.01
cjsp_5_50_10_#2	**49**	0.11	49	180.01
cjsp_5_50_10_#3	**44**	8.61	44	180.01
cjsp_5_50_10_#4	45	180.01	45	180.01
cjsp_5_50_10_#5	**54**	0.03	54	180.01
cjsp_5_50_10_#6	**53**	0.19	53	180.01
cjsp_5_50_10_#7	**59**	0.02	59	180.01
cjsp_5_50_10_#8	47	180.01	47	180.01
cjsp_5_50_10_#9	**45**	0.4	45	180.01
cjsp_5_50_10_#10	**52**	1.9	52	180.01
cjsp_5_50_5_#1	**65**	30.74	65	180.01
cjsp_5_50_5_#2	**70**	16.65	70	180.01
cjsp_5_50_5_#3	87	180.01	83	180.02
cjsp_5_50_5_#4	**83**	2.67	83	180.01
cjsp_5_50_5_#5	**87**	0.04	87	180.01
cjsp_5_50_5_#6	85	180.01	79	180.01
cjsp_5_50_5_#7	80	180.01	77	180.02
cjsp_5_50_5_#8	**72**	1.77	72	180.01
cjsp_5_50_5_#9	**78**	0.25	79	180.01
cjsp_5_50_5_#10	**85**	0.04	85	180.01

Annexes D

TABLE 5.14 – Résultats numériques pour la procédure de Hanen avec $H_{es} = 3$ (Suite)

	$H_{es} = 3$			
	Hanen V2		Hanen V1	
	Solution	Temps	Solution	Temps
cjsp_8_80_4_#1	**141**	0.35	141	180.05
cjsp_8_80_4_#2	**120**	0.42	120	180.01
cjsp_8_80_4_#3	155	180.01	155	180.01
cjsp_8_80_4_#4	**129**	0.47	129	180.08
cjsp_8_80_4_#5	**145**	0.44	145	180.01
cjsp_8_80_4_#6	**121**	0.42	121	180.01
cjsp_8_80_4_#7	115	180.01	115	180.01
cjsp_8_80_4_#8	**124**	0.84	124	180.01
cjsp_8_80_4_#9	**144**	0.43	144	180.01
cjsp_8_80_4_#10	**133**	0.4	133	180.01
cjsp_5_100_10_#1	130	180.01	122	180.01
cjsp_5_100_10_#2	114	180.01	102	180.01
cjsp_5_100_10_#3	146	180.01	138	180.01
cjsp_5_100_10_#4	116	180.01	105	180.01
cjsp_5_100_10_#5	96	180.01	92.5	180.01
cjsp_5_100_10_#6	114	180.01	108	180.03
cjsp_5_100_10_#7	110	180.01	112	180.01
cjsp_5_100_10_#8	114	180.01	103	180.01
cjsp_5_100_10_#9	100	180.01	95	180.01
cjsp_5_100_10_#10	111	180.01	111	180.01
cjsp_5_100_5_#1	**145**	2.11	145	180.01
cjsp_5_100_5_#2	152	180.01	156	180.01
cjsp_5_100_5_#3	**143**	106.01	143	180.01
cjsp_5_100_5_#4	**155**	67.37	155	180.01
cjsp_5_100_5_#5	138	180.01	138	180.01
cjsp_5_100_5_#6	144	180.01	149	180.03
cjsp_5_100_5_#7	179	180.01	179	180.01
cjsp_5_100_5_#8	151	180.01	151	180.01
cjsp_5_100_5_#9	142	180.01	142	180.03
cjsp_5_100_5_#10	**154**	0.68	154	180.01

Table des figures

Liste des tableaux

Bibliographie

[ABAH13] M. Ayala, A. Benabid, C. Artigues et C. Hanen : The resource-constrained modulo scheduling problem : an experimental study. *Computational Optimization and Applications*, 54(3):645–673, 2013.

[ACG92] S. Amar, E. Craye et J.C Gentina : A method of hierarchical specification and prototyping of fms. *Proceedings of the IEEE-ETFA*, 1:44–49, 1992.

[Aya11] Maria Ayala : *Programmation linéaire en nombres entiers pour l'ordonnancement cyclique sous contraintes de ressources.* Thèse de doctorat, Université Paul Sabatier, 2011.

[BBG12] Peter Brucker, Edmund K Burke et Sven Groenemeyer : A branch and bound algorithm for the cyclic job-shop problem with transportation. *Computers & Operations Research*, 2012.

[BCL02] Frédéric Boussemart, G. Cavory et Christophe Lecoutre : Solving the cyclic job shop scheduling problem with linear precedence constraints using CP techniques. *In IEEE International Conference on Systems, Man and Cybernetics(SMC'02)*, Hammamet, Tunisia, oct 2002.

[BCOQ92] F. Baccelli, G. Cohen, G. J. Olsder et J. P. Quadrat : *Synchronization and Linearity.* Wiley, 1992.

[Ber70] Claude Berge : *Graphes et hypergraphes.* Dunod, 1970.

[BFH12] T. BenRahhou, M. Fink et L. Houssin : Une procédure pour la résolution du problème de jobshop cyclique. *In Conférence Internationale Francophone d'Automatique (CIFA2012)*, pages 913–918, Grenoble, France, 2012.

[BH11] A. Benabid et C. Hanen : Worst case analysis of decomposed software pipelining for cyclic unitary rcpsp with precedence delays. *Journal of Scheduling*, 14(5):511–522, 2011.

[BHA10] T. BenRahhou, L. Houssin et C. Artigues : Une approche par la théorie des tas pour le jobshop cyclique. *In International Conference of Modeling and Simulation (MOSIM'10)*, Hammamet , Tunisie, 2010.

[BHF12] T. BenRahhou, L. Houssin et M. Frey : A new procedure for the cyclic job shop problem. *International Journal of Production Research*, 2012. submitted.

[BK05] Peter Brucker et Thomas Kampmeyer : Tabu search algorithms for cyclic machine scheduling problems. *Journal of Scheduling*, 8:303–322, July 2005.

[BK08] P. Brucker et T. Kampmeyer : A general model for cyclic machine scheduling problems. *Discrete Applied Mathematics*, 156(13):2561 – 2572, 2008.

[BLB06] J.-L. Bouquard, C. Lenté et J.-C. Billaut : Application of an optimization problem in max-plus algebra to scheduling problems. *Discrete Applied Mathematics*, 154(15):2064 – 2079, 2006.

[BLBM11] Alessio Bonfietti, Michele Lombardi, Luca Benini et Michela Milano : A constraint based approach to cyclic rcpsp. *Principles and Practice of Constraint Programming–CP 2011*, pages 130–144, 2011.

[BMBV12] C. Bloch, M.-A. Manier, P. Baptiste et C. Varnier : Hoist scheduling problem. *In* P. Lopez et F. Roubellat, éditeurs : *Production scheduling*. WILEY, 2012.

[BP01] J. Y. Le Boudec et Thiran P. : *Network Calculus*. Springer Verlag LNCS 2050, 2001.

[Bra93] H. Braker : *Algorithms and applications in timed discrete event systems*. PhD thesis, Delft University of Technology, Dec 1993.

[Bra08] Nadia Brauner : Identical part production in cyclic robotic cells : Concepts, overview and open questions. *Discrete Applied Mathematics*, 156(13):2480–2492, 2008.

[CC88] J. Carlier et P. Chrétienne : *Problèmes d'Ordonnancement : modélisation, complexité, algorithmes*. Masson, Paris, 1988.

[CC05] Ada Che et Chengbin Chu : A polynomial algorithm for no-wait cyclic hoist scheduling in an extended electroplating line. *Operations Research Letters*, 33(3):274–284, 2005.

[CC09] Ada Che et Chengbin Chu : Multi-degree cyclic scheduling of a no-wait robotic cell with multiple robots. *European Journal of Operational Research*, 199(1):77–88, 2009.

[CCP95] Haoxun Chen, Chengbin Chu et Jean-Marie Proth : Cyclic hoist scheduling based on graph theory. *In IEEE*, pages 451–459, 1995.

[CDG05] Guillaume Cavory, Rémy Dupas et Gilles Goncalves : A genetic approach to solving the problem of cyclic job shop scheduling with linear constraints. *European Journal of Operational Research*, 161(1):73–85, 2005.

[CdK97] Y. Crama et J. Van de Klunder : *Robotic flowshop scheduling is strongly NP-complete*. Ten Years LNMB, 1997.

[CDQV83] G. Cohen, D. Dubois, J. P. Quadrat et M. Viot : Analyse du comportement périodique des systèmes de production par la théorie des dioïdes. Rapport de recherche 191, INRIA, Le Chesnay, France, 1983.

[CDQV85] G. Cohen, D. Dubois, J. P. Quadrat et M. Viot : A linear system theoretic view of discrete event processes and its use for performance evaluation in manufacturing. *IEEE Trans. on Automatic Control*, AC–30:210–220, 1985.

[Chr00] Philippe Chrétienne : On Graham's bound for cyclic scheduling. *Parallel Computing*, 26(9):1163–1174, 2000.

[CKvdKL00] Y. Crama, V. Kats, J. van de Klundert et E. Levner : Cyclic scheduling in robotic flowshops. *Annals of Operations Research : Mathematics of Industrial Systems 96*, pages 97–124, 2000.

[CLRS09] T. H. Cormen, C. E. Leiserson, R. L. Rivest et C. Stein : *Introduction to Algorithms*. The MIT Press, 2009.

[CMQV89] G. Cohen, P. Moller, J. P. Quadrat et M. Viot : Algebraic tools for the performance evaluation of discrete event systems. volume 77, Jan. 1989.

[Coh95] G. Cohen : *Théorie algèbrique des systèmes à événements discrets*. Polycopié de cours de l'École des Mines de Paris, 1995.

[CTCG+98] J. Cochet-Terrasson, G. Cohen, S. Gaubert, M. Mc Gettrick et J. P. Quadrat : Numerical computation of spectral elements in max-plus algebra. *In Proceedings of the IFAC Conference on System Structure and Control*, Nantes, 1998.

[DA89] R. David et H. Alla : *Du Grafcet aux réseaux de Petri*. Hermès, Paris, 1989.

[DAA08] B. Dupont de Dinechin, C. Artigues et S. Azem : Resource-constrained modulo scheduling. *In* C. Artigues, S. Demassey et E. Néron, éditeurs : *Resource-Constrained Project Scheduling. Models, Algorithms, Extensions and Applications*. ISTE-WILEY, 2008.

[DG98] Ali Dasdan et Rajesh K. Gupta : Faster maximum and minimum mean cycle algorithms for system-performance analysis. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 17(10), 1998.

[DGSS05] Milind Dawande, H. Neil Geismar, Suresh P. Sethi et Chelliah Sriskandarajah : Sequencing and scheduling in robotic cells : Recent developments. *Journal of Scheduling*, 8:387–426, October 2005.

[Die10] Reinhard Diestel : *Graph theory*, volume 173. Springer-Verlag, 4 édition, 2010.

[Dup04a] R. Dupas : *Amélioration de performance des systémes de production : apport des algorithmes évolutionnistes aux problémes d'ordonnancement cycliques*. Thèse de doctorat, HDR, Université d'Artois, 2004.

[Dup04b] B. Dupont de Dinechin : From machine scheduling to VLIW instruction scheduling. *ST Journal of Research*, 1(2), 2004.

[Dup07] B. Dupont de Dinechin : Time-indexed formulations and a large neighborhood search for the resource-constrained modulo scheduling problem. *In* P. Baptiste, G. Kendall, A. Munier-Kordon et F. Sourd, éditeurs : *3rd Multidisciplinary International Scheduling conference : Theory and Applications*, pages 144–151, 2007.

[ED97] A. Eichenberger et E.S. Davidson : Efficient formulation for optimal modulo schedulers. *In SIGPLAN - PLDI'97*, 1997.

[FBH12] M. Fink, T. BenRahhou et L. Houssin : A new procedure for the cyclic job shop problem. *In IFAC Symposium on Information Control Problems in Manufacturing (INCOM 2012)*, Bucarest , Roumanie, 2012.

[FCP03] J.W. Herrmann F. Chauvet et J. M. Proth : Optimization of cyclic production systems : a heuristic approch. *IEEE Trans. on Robotics and Automation*, 19:150–154, 2003.

[GM79] M. Gondran et M. Minoux : *Graphes et algorithmes*. Eyrolles, Paris, 1979. Engl. transl. *Graphs and Algorithms*, Wiley, 1984.

[GM95] Michel Gondran et Michel Minoux : *Graphes et algorithmes, 3e édition revue et augmentée*. Eyrolles, 1995.

[GM99] S. Gaubert et J. Mairesse : Modeling and analysis of timed petri nets using heap of pieces. *IEEE Trans. on Automatic Control*, 44(4), Apr. 1999.

[GS94] Franco Gasperoni et Uwe Schwiegelshohn : Generating close to optimum loop schedules on parallel processors. *Parallel Processing Letters*, 4:391–403, 1994.

[Han90] C. Hanen : Les tables de réservation numériques : un outil pour la résolution de certains problèmes d'ordonnancement cycliques. *Revue franÃ§aise d'automatique, d'informatique et de recherche opérationnelle*, pages 97–122, 1990.

[Han94] C. Hanen : Study of a np-hard cyclic scheduling problem : The recurrent job-shop. *European Journal of Operational Research*, 72(1):82 – 101, 1994.

[HKDG08] Tiente Hsu, Ouajdi Korbaa, Rémy Dupas et Gilles Goncalves : Cyclic scheduling for fms : Modelling and evolutionary solving approach. *European Journal of Operational Research*, 191(2):464–484, 2008.

[HLB04] L. Houssin, S. Lahaye et J. L. Boimond : Modelling and control of urban bus networks in dioids algebra. *In Proceedings of WODES'2004*, Reims, France, 2004.

[HLB12] L. Houssin, S. Lahaye et J.-L. Boimond : Control of (max,+)-linear systems minimizing delays. *Discrete Event Dynamic Systems*, 2012.

[HM95a] C. Hanen et A. Munier : Cyclic scheduling on parallel processors : an overview. *In* P. Chretienne, E.G. Coffman, J.K. Lenstra et Z.Liu, éditeurs : *Scheduling Theory and its Applications*, pages 194–226. John Wiley and Sons, 1995.

[HM95b] C. Hanen et A. Munier : *Scheduling Theory and Its Applications*, chapitre Cyclic Scheduling on Parallel Processors : An Overview. Wiley, 1995.

[HM95c] C. Hanen et A. Munier : A study of the cyclic scheduling problem on parallel processors. *Discrete Applied Mathematics*, 57:167–192, 1995.

[HM09] Claire Hanen et Alix Munier : Periodic schedules for linear precedence constraints. *Discrete Applied Mathematics*, 157(2):280–291, 2009.

[Hou11] L. Houssin : Cyclic jobshop problem and (max,plus) algebra. *In Proceedings of IFAC WC*, Milan, Italy, aug. 2011.

[How60] R.A. Howard : *Dynamic programming and Markov processes.* Technology Press of Massachusetts Institute of Technology, 1960.

[HP89] H. P. Hillion et J. M. Proth : Performance evaluation of job-shop systems using timed event graphs. *IEEE Transaction on Automatic Control*, 34(1):3–9, Jan. 1989.

[IS95] I. Ichimori et F. Soumis : Schedule efficiency in a robotic production cell. *International Journal of Flexible Manufacturing Systems*, 7:5–26, 1995.

[Kam06] Thomas Kampmeyer : *Cyclic Scheduling Problems.* Thèse de doctorat, Universitat Osnabruck, 2006.

[Kar78] R. M. Karp : A characterization of the minimum cycle mean in a digraph. *Discrete Math*, 23, 1978.

[KCG02] O. Korbaa, H. Camus et J.-C. Gentina : A new cyclic scheduling algorithm for flexible manufacturing systems. *International Journal of Flexible Manufacturing Systems*, 14(2):173–187, 2002.

[Ken99] Yura Kenji : Cyclic scheduling for re-entrant manufacturing systems. *International Journal of Production Economics*, 60-61:523–528, 1999.

[KG00] O. Korbaa et J.C. Gentina : Cyclic scheduling in flexible manufacturing systems. *Revue internationale d'ingénierie des systèmes de production mécanique*, pages 47–54, 2000.

[KL97] V. Kats et E. Levner : A strongly polynomial algorithm for no-wait cyclic robotic flowshop scheduling. *Operations Research Letters*, 21:171–179, 1997.

[KLB09] J. Komenda, S. Lahaye et J.-L. Boimond : Supervisory control of (max,+) automata : A behavioral approach. *Discrete Event Dynamic Systems*, 19(4): 525–549, 2009.

[KO81] R. M. Karp et J. B. Orlin : Parametric shortest path algorithms with an application to cyclic staffing. *Discrete Applied Mathematics*, 3:37–45, 1981.

[LK98] E. Levner et V. Kats : A parametrical critical path problem and an application for cyclic scheduling. *Discrete Applied Mathematics*, 87:149–158, 1998.

[LKdC10] E. Levner, V. Kats, A. L. de Pablo et D. Cheng : Complexity of cyclic scheduling problems : A state-of-the-art survey. *Computers & Industrial Engineering*, 59:352–362, 2010.

[LMQ05] P. Lotito, E. Mancinelli et J.-P. Quadrat : A minplus derivation of the fundamental car-traffic law. *IEEE Trans. on Automatic Control*, 50(5):699–705, May 2005.

[LW89] L. Lei et T.J. Wang : A proof : the cyclic hoist scheduling problem is NP-complete. Rapport technique, School of Management, Rutgers University, New Jersey USA, 1989.

[MT02] M. Middendorf et V.G. Timkovsky : On scheduling cycle shops : classification, complexity and approximation. *Computers & Industrial Engineering*, 5:135–169, 2002.

[Mun91] Alix Munier : Résolution d'un problème d'ordonnancement cyclique à itérations indépendantes et contraintes de ressources. *RAIRO. Recherche opérationnelle*, 25:161–182, 1991.

[Mun96] A. Munier : The complexity of a cyclic scheduling problem with identical machines and precedence constraints. *European journal of operational research*, 1996.

[Mur89] T. Murata : Petri Nets : Properties, Analysis and Applications. *IEEE Proceedings : Special issue on Discrete Event Systems*, 77:541–580, Jan. 1989.

[NYO91] R.E. Tarjan N.E. Young et J.B. Orlin : Faster parametric shortest path and minimum-balance algorithms. *Networks*, 21:205–221, 1991.

[OCG97] Korbaa Ouajdi, H. Camus et Jean-Claude Gentin : Heuristic for the resolution of the general FMS cyclic scheduling problem. *In Systems, Man, and Cybernetics. Computational Cybernetics and Simulation. IEEE Conference*, pages 2903–2908, 1997.

[Pin05] M. Pinedo : *Planning and Scheduling in Manufacturing and Services*. Springer, 2005.

[PU76] L. W. Phillipsa et P. S. Ungerb : Mathematical programming solution of a hoist scheduling program. *IIE Transactions*, 8:219–225, 1976.

[RH08] P. Richard et C. Haro : Applications des réseaux de pétri. *IEEE*, 2008.

[Rou92] Robin Roundy : Cyclic schedules for job shops with identical jobs. *Mathematics of Operations Research*, 17:842–865, September 1992.

[RV10] Y. Robert et F. Vivien, éditeurs. *Introduction to scheduling*. CRC Press, 2010.

[SL02] Jeong-Won Seo et Tae-Eog Lee : Steady-state analysis and scheduling of cyclic job shops with overtaking. *International Journal of Flexible Manufacturing Systems*, 14:291–318, 2002.

[XHG02] Jian Xu, Tienté Hsu et Gilles Goncalves : A genetic algorithm to solving the problem of flexible manufacturing system cyclic scheduling. *In Systems, Man and Cybernetics, 2002 IEEE International Conference*, pages 640–651, 2002.

Synthesis Algorithms for Cyclic Scheduling Problems

Abstract

Research on scheduling has been mobilizing a large number of researchers, this is mainly due to the broad range of application of scheduling problems, among them, there is the problem of tracking multiple workshops, commonly called "JobShop". JobShop problems can often be simplified by considering them as cyclic problems. The tasks scheduling becomes cyclic and its purpose is to organize the production activities by repeating a basic cycle that has been optimized. Many parameters are involved in the optimization of the basic cycle such as the period of the cycle chosen, the order of operations to perform basic work, the duration of these operations, the number of outputs per cycle, etc.

Several approaches have been used to solve this problem, among which the approach by Petri nets, especially in timed event graphs, graph approach, linear programming approach and heap of pieces approach.

The graph approach allows a graphical representation of the problem as a graph where the nodes represent the different operations and where the arcs illustrate constraints. A cyclic scheduling problem has a feasible solution if and only if it is consistent. This consistency property of a cyclic scheduling problem and its associated graph allows to prune the search tree of the branch-bound procedure proposed for this approach.

In the heap of pieces approach, the sub-problem of evaluating a solution can be easily solved. Indeed, by translating the problem into an appropriate mathematical structure, finding the cycle time is equivalent to the calculation of an eigenvalue of a product of matrices where each matrix represents an elementary operation. This property is particularly significant in the case of successive sets of evaluation of many schedules. In addition, the theory allows an intuitive representation of scheduling, since it is illustrated as a stack of several blocks. Several cycles will be symbolized by the stack of piles.

Keywords

Cyclic scheduling, Max-plus algebra, Discrete optimization, Heap of pieces, Graph theory.

Auteur : Touria CHAFQANE BEN RAHHOU
Directeur de thèse : Laurent HOUSSIN
Lieu et date de soutenance : Toulouse, le 24 juin 2013
Discipline Administrative : Génie Industriel (4200046)

Nouvelles Méthodes pour les Problèmes d'Ordonnancement Cyclique.

Résumé

Les travaux de recherche concernant l'ordonnancement mobilisent un nombre important de chercheurs. Cette forte émulation est principalement due au large panorama des problématiques d'ordonnancement. Parmi elles, le problème d'atelier à cheminement multiple, communément appelé « Job-Shop », tient une place particulièrement prépondérante tant ce problème est rencontré dans le milieu industriel. De nombreux sujets de recherche, en France et à l'étranger, sont issue de cette problématique.

Les problèmes de Job-Shop peuvent souvent être simplifiés en les considérant comme des problèmes cycliques. L'ordonnancement des tâches devient ainsi cyclique et son objectif est d'organiser les activités de production en répétant un cycle de base que l'on a optimisé. De nombreux paramètres entrent en jeu dans l'optimisation du cycle de base tels que la période du cycle choisie, l'ordre des opérations élémentaires pour réaliser un travail, la durée de ces opérations, le nombre de produits à réaliser par cycle, etc.

Plusieurs approches ont été utilisées pour résoudre ce problème. Parmi elles, nous pouvons citer l'approche par réseaux de Petri et plus particulièrement par graphes d'événements temporisés, l'approche par les graphes, l'approche par la programmation linéaire et l'approche par la théorie des tas.

Dans cette thèse, plusieurs approches sont étudiées et comparées pour la résolution de problèmes d'ordonnancement cycliques avec pour objectif la minimisation du temps de cycle.

Mots-clefs

Ordonnancement cyclique, Algèbre max-plus, Optimisation discrète, Théorie des tas, Théorie des graphes.

Université Toulouse III - Paul Sabatier
118 Route de Narbonne
31 062 Toulouse cedex 9

Ĺaboratoire d'Analyse et d'Architecture des Systèmes
7, avenue du Colonel Roche
31 077 Toulouse Cedex 4